Interface

2004 10

Contents



[表紙デザイン：㈱プランニング・ロケッツ]





特集

用途に合わせたさまざまなUSB機器を自在に設計するための

USBホスト&ターゲット・システム設計技法

Cover Story The design method of USB host and the target system 51

Interface

2004 10

Contents

話題のテクノロジ解説

ショウ・レポート&コラム

一般解説&連載

情報のページ

連載「TOPPERSで学ぶRTOS技術」,「はじめて使うμClinux」と「VxWorksを使ったRTOS技術の基礎と応用」は,お休みさせていただきます.

▶ 組み込み技術の総合展示会

第７回 組込みシステム開発技術展 ESEC

北村 俊之

組み込みシステムの応用技術が一堂に会する展示会「ESEC」が７月７日(水)～９日(金)の３日間，東京ビッグサイトで開催された(写真１)．主催はリード エグジビジョン ジャパン(株)．今回で第７回目を迎える本展示会は，出展社数も約340社と，昨年を上回る過去最大規模での開催となった．今回は展示会場全体が，「ボード・コンピュータ」，「タッチパネル」，「組込みLinux」，「ユビキタス・ネットワーク」，「設計・開発サービス/コンサルティング」の五つのゾーンに分けられていた．



写真１　会場入り口のようす

専門セミナでは，「ユビキタス時代到来！ウィンドウズ，トロンのトップが語る！！」をテーマに，古川 享(マイクロソフト)，坂村 健(東京大学)両氏のキーノート・セッションを始めとした合計20のセミナが開催され，こちらも盛況だった．

展示会全体としても，例年同時開催されている「第13回 ソフトウェア開発環境展(SODEC)」，「第９回 データウェアハウス＆CRM EXPO」，「第６回データストレージEXPO」に加え，今年から新たに「第１回 情報セキュリティEXPO(i-Security)」が開催されるなど，過去最大規模での開催となっていた．今回は，「第７回 組込みシステム開発技術展(ESEC)」および「第１回 情報セキュリティEXPO(i-Security)」を中心にレポートを行う．

● 第７回 組込みシステム開発技術展 ESEC）

アドバネットは，Pentium M搭載のCompactPCI CPUボード「A6pci8014」，ULV Celeron搭載のCompactPCI CPUボード「A6pci8016」(写真２)，分散型コントローラ「Adbc7005」，JPEG圧縮ボード「Adpmc2211」，産業用コンピュータ・システム「Aicos」など，多彩な製品ラインナップで来場者の注目を集めていた．「Aicos」は，Intel ULV Celeron(650MHz)プロセッサを搭載しており，本体にCompactFlash，10/100Base-T，PCMCIA Type II，VGA，PS/2，USBなどのペリフェラルを装備し，スタンドアロンPCとしても利用が可能であるという．



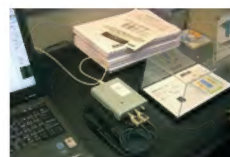
写真２　アドバネットのA6pci8016

ソリトンシステムズ(写真３)は，マルチ・インターフェース・サーバ「TCS-8000/8010シリーズ」(東亜ディーケーケー)を展示していた．同製品は，OSにLinuxを採用し，RS-232-C，Ethernet，USB，CompactFlashの各種インターフェースを装備した小型コンピュータという位置付けになるという．低速データから高速データまでの取り込みが可能で，データはCompactFlashまたはUSB対応のストレージなどへの蓄積が可能となっている．蓄積したデータは，有線/無線LAN，DoPa，モデムなどで転送できる．



写真３　ソリトンシステムズのブース

アドテックシステムサイエンスは，USB規格に準拠した，ディジタル・ストレージ・オシロスコープ「ASB-3000シリーズ」(写真４，参考出品)の展示デモを行っていた．同製品は，小型軽量ながら，アナログ周波数帯域幅40MHz，サンプリング・レート100Msps(２チャネル同時)を実現しており，USBポート搭載のノートPCなどに接続することで，携帯性に優れた測定器として使用することが可能となっている．実際の製品の発売に際しては，展示品よりさらに小型化する予定だという．


写真４　アドテックシステムサイエンスのASB-3000シリーズ

MOXA TECHNOLOGIES(写真５)は，シリアル-Ethernet変換ソリューション「NPort 5000シリーズ」，エンベデッド・ネットワーク「NE-4000シリーズ」，ユニバーサル・コミュニケータ「UC-7400シリーズ」，ワイヤレス・シリアル・デバイス・サーバ「NPort W2000」など，多彩な製品を展示していた．「UC-7400シリーズ」は，XScaleコアのIXP-422 266MHzプロセッサを搭載し，８ポート・シリアル，Ethernet×２，USB，PCMCIA，CompactFlashインターフェースを装備したRISCベースのコミュニケーション・サーバ．MontaVista Linuxをプリインストールしているため，GNUクロスコンパイラを使用することで，デスクトップPC用に書かれたソフトウェアをソース・コードの変更なしに同製品にエクスポート可能とのことである．



写真５　MOXA TECHNOLOGIESのブース

日本テクトロニクスは，任意波形ジェネレータ「AWG710B型」「AWG615型」，ディジタル・ストレージ・オシロスコープ「TDS6000Bシリーズ」の展示を行っていた．「AWG710B型」(写真６)は4.2Gsps，「AWG615型」は2.7Gspsの高速サンプリングによるアナログ信号出力をサポートし，２台の同期動作で２チャネルのアナログ出力と４チャネルのマーカを可能にしているという．「TDS6000Bシリーズ」は，４チャネル同時の最高20Gspsサンプル・レートと，各チャネル最大32Mポイントのレコード長を実現した，アクイジション・アーキテクチャの装備を大きな特徴としている．


写真６　日本テクトロニクスのAWG710B型とTDS6804B

● 第１回 情報セキュリティ EXPO（i-Security）

三和コムテックは，Webサイト・セキュリティおよび個人情報保護を目的としたサービス「HACKER SAFE」(写真７)の紹介を行っていた．同サービスでは，自動リモート・スキャンにより，Webサイトのセキュリティ上の脆弱性の有無を毎日検査し，安全性を証明する．VISA，MasterCard，American Expressが義務付けているネットショップ・セキュリティの必須条件を満たしており，検査が免除され，第三者専門機関(HACKER SAFE)が毎日検査し，証明することで高い信頼性を保証している．すでに全世界で55,000以上のサイトが導入実績を持つとのことだった．



写真７　三和コムテックのHACKER SAFE

プログレッシブ・システムズは，統合映像監視ソリューション「UniArgus」の展示紹介を行っていた．同製品は，監視映像のモニタリング，入室管理の設定，アラームの設定，カメラなど監視デバイスの管理・設定など，監視システム全体を遠隔地からコントロールできることを特徴としている．用途に応じて家庭用，小規模オフィスから大規模オフィス用まで幅広いプラットホームをサポートしている．

第6回 組込みシステム技術に関するサマーワークショップ SWEST6

■日時：2004年7月22日(木)～23日(金)
■場所：遠鉄ホテルエンパイヤ(静岡県浜松市)

現場で働く組み込み技術者向けに毎年1回開催されている「組込みシステム技術に関するサマーワークショップ SWEST6」が浜松湖畔で開催された．今年で第6回を迎えるSWESTだが，今回はメンバによる小型ロケットの打ち上げが行われるなど，充実した内容であった．

宇宙航空研究開発機構(JAXA)奥田 一実氏の基調講演「宇宙機搭載のソフトウェア」は，非常に高い保証が求められるクリティカル・システムである宇宙機に関する講演だった．JAXAの宇宙機開発においては，無人機は二重，有人機は三重のバックアップ体制で故障時に備えるとのことだった．また，チャレンジャー号の事故以降のNASAの対策である「ソフトウェア独立検証及び有効性確認(IV&V；Software Independent Verification and Varidator)」が紹介された．これは独立した査察機関にソフトウェアの検証・評価を行わせるというもので，これにより高い信頼性を保証するというシステムである．また，フリーなμITRON実装であるTOPPERSの宇宙用耐放射線性MPU(MIPSアーキテクチャ)への移植を開始し，宇宙機への搭載をめざすことも発表された．

その後，会場を移し，ポスター・セッションが行われた．(株)ヴィッツのTOPPERS/OSEKは，ヨーロッパの車載用OS，OSEKをTOPPERSプロジェクトにて実装したものである．展示デモでは車の模型を動かしていたほか，ゲームボーイアドバンスへも移植し，注目を集めていた．

二日目の早朝には「組み込みシステム開発のリアルな教育教材を開発しよう」をスローガンとした「サーベイヤ計画」の一環として，模型ロケット「Hamana1」の打ち上げが行われた．ロケットにはGPSが搭載され，AVRマイコンを使って取得した位置データをEEPROMに書き込むことにより，航路を記録するというシステムになっている．打ち上げは無事成功し，データの取得も行えた．

午後のチュートリアルでは，サーベイヤ計画代表の二上 貴夫氏(（株）東陽テクニカ)により，同計画の趣旨説明と結果報告が行われた．その中で二上氏は「組み込みシステムはソフトやハードの製作だけでなく，実環境でのテストや雑務なども含めたトータルなものであり，しかも実際には思いもかけないトラブルが発生する．これらも含めて管理するのが組み込み開発である」とし，組み込み開発ということばの意味する幅広さを強調していた．同時にGPSの取得データも公開され，水平方向に関してはデータの取得に成功し軌跡を表示できたが，高さ方向に関してはGPSが高さ方向への高速な移動に追従せず，正確なデータが取得できなかったとしていた．

分科会「プロセッサアーキテクチャとリアルタイムシステム～ハードリアルタイムシステムにキャッシュを使うの？～」は，冨山 宏之氏(名古屋大学)と高野 裕之氏(東芝セミコンダクター)をコーディネータに座談会形式で行われた．近年の32ビットRISC CPUにはキャッシュが搭載されており，性能向上のために使用したいのだが，ハードリアルタイム・システムにおいては最悪実行時間(WCAT)の見積もりが難しくなるという問題が存在する．これを回避するためには，キャッシュのプリフェッチであらかじめすべてのコードとデータをキャッシュに置いてしまう，キャッシュよりも高速な内部SRAMなどに置いたほうが予測がしやすいなど，さまざまな角度からの発言があった．


JAXA 奥田 一実氏



基調講演のようす


ロケットの調整に余念のない二上氏


打ち上げのようす


(株)ヴィッツのTOPPERS/OSEKのデモ

ボーランド，コードとモデリングを統合した開発環境「Borland Together Edition for Microsoft Visual Studio .NET 2.0」を発売

■日時：2004年7月6日(火)
■場所：新宿ファーストウエスト(東京都新宿区)

ボーランド(株)は，Microsoft Visual Studio .NETにUMLモデリング機能を追加するBorland Together Edition for Microsoft Visual Studio .NETを￥25,200で発売した．同製品はUMLモデリング機能を追加するだけでなく，ソース・コードとモデル図を自動的に同期することが特徴である．これは，ソース・コードを修正するとモデル図が自動的に修正され，逆にモデル図を修正するとソース・コードへと修正が自動的に反映され，ソース・コードとモデル図の不一致といった事態を防止することができる．対応する言語はC#とVB.NET．

また，モデル図を生成する際にソース・コードの解析を行っているが，これを用いた機能として，ソース・コードの中で未使用の変数，命名規則にしたがっていない部分，コーディング・ミスが疑われる部分など，エラーが起こりそうな箇所を指摘するQA機能もサポートしている．同様に既存のソース・コードを解析するリファクタリングも行える．

ハッカーの常識的見聞録



今月の常識

2005年 Itanium2もDDR2，PCI-Expressに

広畑 由紀夫

☆この夏デスクトップ向けにようやく姿を見せ始めたDDR2，PCI-X対応チップセットが2005年にはItanium2にも搭載されてきます．今回はこの次期Itanium2に触れてみたいと思います．

● 次期Itanium2用第3世代チップセット，開発コード「Bayshore」

まず，バス速度については，システム間転送が現行の400MHzから667MHzへと向上し，長期安定性と速度の向上が図られるようです．また，システム・バスのクロック向上に従って，現在のItanium2における動作速度1.5GHz（デュアル・プロセッサ向けに1.6GHz版製品あり）から，少なくとも2GHzには向上するのではないかと思われます．そして，DDR2プロトコルのサポートで，より高速な転送レートを確保し，さらにPCI-Expressへ移行することによってレガシを切り離して負荷を軽減し，回路の簡素化などといったシステム全体の向上が図られています．

Intel 9xxチップ・セット・シリーズにて採用されたDDR2およびPCI-Expressですが，2004年7月現在では，まだまだ性能を生かせているといいきれません．しかし，DDR2で実装されているメモリのレイテンシの向上などは，これから期待できる部分です．

● 次期Itanium2，開発コード「Montecito」

現在発表されている情報によると，Montecitoで期待できる強化部分は次に示すようになるようです．

① デュアルコアにより，1プロセッサあたり，4スレッド対応マルチスレッド動作
② システム・バス速度の向上による各システム間データ転送速度の向上
③ ピーク時に動作速度を向上させる「Foxton」
④ 低負荷時に消費電力を落とす「SpeedStep」
⑤ 24Mバイトの大容量L3キャッシュ

これらの強化機能のうち，筆者が興味をもっているのは「Foxton」です．「Foxton」によってピーク時動作をいかにしてやりすごし，情報処理の延滞を減らしつつ安定性を保つのかをいずれ見てみたいと考えています．ハイパ・スレッディングや「Speed Step」など，すでにPentium-MやXeonにおいて実装されている技術のItanium2への投入も興味深いことです．Xeonのハイパ・スレッド強化やデュアルコア化の発表にしたがい，Itanium2への移行が遅れぎみともいわれますが，Xeonより単一プロセッサの処理が強化されたItanium2は，今後ハイエンド・サーバ領域などにおいて有効な選択肢となるでしょう．

● ソフトウェアの対応

プロセッサがどれほど強化されたとしても，まずはOSが動作しなければ使えません．現在，Itanium2用としてはWindows Server 2003，HP-UX，RedHatなどが対応し，Windows Server 2003にはSP1（7月現在はβ版）でIA 32-ELが同封予定とされています．IA-64アーキテクチャのみを利用したサーバ・システムという面からすると，すでに多くのサーバ・システムが移植され，バックボーン・システムとしてはずいぶんと使いやすくなったと思われます．

今後，ただ単一の64ビット・サーバというだけではなく，IA 32-ELのように，膨大なソフトウェア資産を有効利用することも，より柔軟になってくるかと思われます．筆者が期待するのは，より強力な仮想化技術や，Virtual Server 2005のIA 64版など，Itanium2のもつ安定性や強力な演算性能を使用した従来のサーバ・システムの再統合と仮想化技術です．

● 「Montecito」の先のItanium2

すでに「Montecito」より先のItanium2である「Tukwila」では，デュアルコアからマルチコアへと進化をはじめ，4～16コアで開発される目標とのことです．「Tukwila」の製品発表は2007年予定となっていますが，それまでに現在のシングルコアがマルチコア前提のデュアルコアへの進化がどんどん進むのではないかと筆者はみています．

● 一般のユーザが受ける恩恵は？

サーバ・システムがより高速かつ堅牢に動作するようになることで，現在のデスクトップPCなどでのリアルタイム通信サービスの幅が広がってくることは当然予測されます．近年の急速な通信速度やプロトコルの改善によってMMORPGなどが現実のものになってきていますが，一つのワールド・サーバで5,000人程度を超える接続では不具合が発生しやすいようです．

今後，サーバ側の進歩によってこれらが解消され，数万人～数十万人が同一のワールドを共有した現在の大規模MMORPGを超えたサーバ・ウェアの登場，そして現在よりもより柔軟でリアルタイム性の高い高度なサービスが実現するのではないか考えます．

筆者は，今年の9月7日～9日にサンフランシスコで開かれるIntel Developer Forumにて，より詳細な発表がなされるのではないかと期待しています．

● Intel社の関連情報サイト
http://www.intel.com/pressroom/archive/releases/20040218corp.htm

ひろはた・ゆきお　OpenLab.

IPパケットの隙間から

突然やってきたUUCP接続の終焉

祐安 重夫

土曜日の夕方，仕事をしていたら，突然，背後からバチバチというノイズが聞こえだした．てっきり25年以上も前から使っているラジカセが故障したのかと思って背後にまわってみたら，なんと1台のコンピュータのファンから火花が出ていた．慌てて電源コードを引き抜いた．

どうやらファンが停止して電源に過負荷がかかり，コンデンサが燃えたらしい．もう少ししたら仕事を終わらせて，行きつけの神田の寿司屋まで酒でも呑みに出かけようと思っていたところだったので，故障がもう少し遅ければ火事になっていたかもしれない．あぶないところだった．

このマシンは，それまで使用していたソニーのNEWSが故障したため，はじめてワークステーションではなく，PCでUNIX系OSを本格的に動かしはじめたものだった．購入したのは1995年3月のことで，さすがに10年を前にして寿命がつきてしまった．

もちろん，現在まで本格的に使用していたわけではなく，緊急時のバックアップ用として稼働させておいたものである．OSはBSD/OSだった．その当時，PCで動作するBSD系のOSの中では，フリーではないが，それに近い感覚で使用でき，安定した商用OSとしては安価に導入できたのがBSD/OSだったのだ．

現在では，メインのサーバや作業用のマシンは，Linuxに移行してしまったが，今回壊れたマシンの後継機として1999年5月から使用しているBSD/OSマシンはまだ現役で，このPCだけ個人で契約しているADSLのルータをゲートウェイにして，Proxyサーバやプロバイダ側にあるメール・サーバからメールを取得してくるだけのローカルなメール・サーバ，LAN内部向けのDNSなどに使用している．

外部のネットワークへの接続を行ったのは1987年のことで，ソニーのNEWSを導入したのを機に，JUICEとUUCP接続を行ったのが最初である．JUICEはJUNETと相互接続していたし，当時INETクラブと呼ばれていた海外とのゲートウェイも利用できたので，現在と同様にインターネット・メールやネット・ニュースが使用できた．

その後，1993年になって筆者の会社で独自ドメインを取得し，まだ数社しかなかったISPの一つであったIIJと，UUCPの接続契約を行った．JUICE接続時と同様の環境でそのまま移行できた．それからしばらくしてNEWSが故障してしまい，1995年になってPC UNIXに移行したというわけである．

この時に，1994年に取得だけしておいたor.jpドメインも，未接続のままで有効期限が切れる寸前になっていたのを，同様にUUCP接続契約をして有効にした．もっとも，会社で取得したco.jpドメインはその後IP接続に移行したし，さらに別に取得した汎用jpドメインは，当然最初からIP接続にした．この時代になると，そもそも新規でUUCP接続ができるISPなど，すでにほとんど存在しなかったし，UUCPで接続する必然性はまるでなかった．

だが，or.jpドメイン一つだけは，滅多に来ないメールを受信する以外に必要がなかったので，UUCPのままになっていた．さすがに新規の契約はもう受け付けてくれなかったが，顧客が残っている以上，そう簡単にUUCPサービスを停止するわけにはいかなかったのだろう．

ところが本年の3月号と4月号の本コラムで書いたような大量のエラー・メールの受信という事態が起きて，こちら側がUUCPから本格的に手を切ることを考えざるをえなくなった．そう思っていたところにIIJから郵便が来て，ついにUUCPサービスを終了すると通告してきた．時代の趨勢(すうせい)からすれば，当然の措置といわなければならないだろう．

現状でUUCPが有効な環境は，IP接続されたホストどうしでUUCP over IPを利用して，ネット・ニュースの配送を行うことくらいだろう．IP上のネット・ニュース配送プロトコルであるNNTP(Network News Transfer Protocol)も存在するが，実はUUCP over IPを利用してUUCPでニュースの配送を行うと，通常はニュース本体を圧縮して転送してくれるので効率が良いのだ．

ところで，IIJの代替サービス案でメール・サーバとDNSをISP側に持ってもらうサービスに移行すると，今まで月に2000円を少し超えるくらいだった料金が，一挙に一桁上がってしまう．これではとても移行するわけにはいかない．逆にいえば，よく今までこんなに儲からないサービスを続けてくれたと感謝すべきなのかもしれないが．

どちらにしても，こうなるとISPの変更という手しかない．幸いなことに，大手のパソコン通信を母体として発展してきたISPが，今までUUCPに支払ってきた料金と大差のない価格で，独自ドメインのホスティング・サービスを始めるようである．登録できるメール・アドレスが少ないとか，利用できるWebスペースが小さいという制限はあるが(もちろんどちらも拡大できるが，その分料金は高くなる)，このor.jpドメインにとっては十分である．

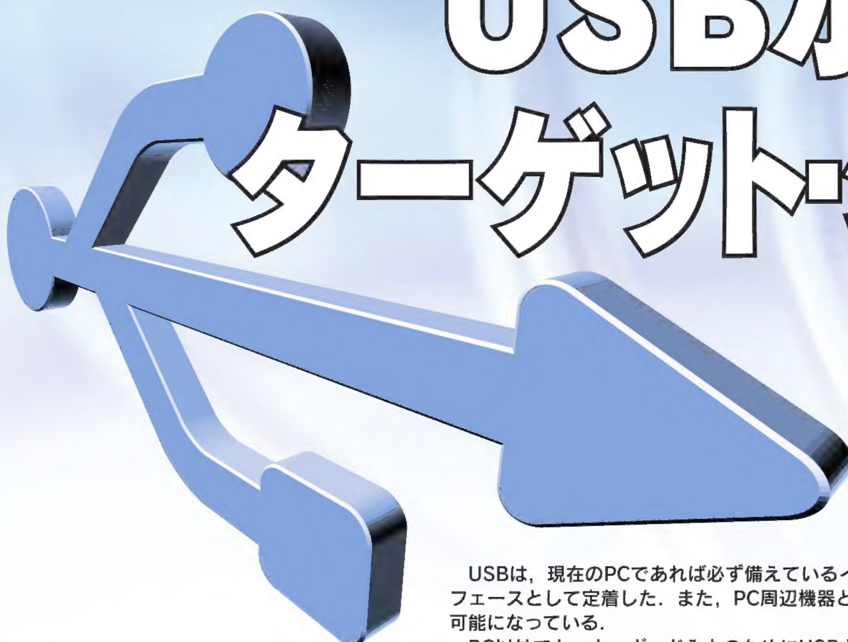
さて，結論が一応でたので，筆者はこれから酒でも呑みに出かけることにしよう．

すけやす・しげお　インターメディア・アクセス

特集

用途に合わせたさまざまなUSB機器を自在に設計するための

USBホスト&ターゲット・システム設計技法



USBは，現在のPCであれば必ず備えているインターフェースと考えても良いほど，標準的なインターフェースとして定着した．また，PC周辺機器として一般的なものは，ありとあらゆるものがUSBで接続可能になっている．

PC以外でも，キーボード入力のためにUSBキーボードが使えるAV機器や，外付けHDDをUSBで接続して容量を増やせるHDDビデオ・レコーダなども登場している．

そして現在では，これまでPCの周辺機器として接続されてきたディジタル・カメラやプリンタが，PCを介さずとも接続可能になり写真を印刷できるようになった．また，PDAにUSB周辺機器を接続したり，PDAどうしを直接USBで接続できるようにもなってきた．

本特集では，まずUSBシステムを正しく理解するために，USBコントローラの分類やWindowsにおけるUSBプロトコル・スタックの構造などを解説し，組み込み機器におけるUSBソフトウェアがどのような構造になっているかを理解する．

さらに，USBターゲット・デバイス，USBホスト・デバイス，そしてOn-The-Go対応デバイスの3種類について，いくつかデバイスを取り上げて解説する．

最後に，USB機器デバッグ方法についてや，PictBridgeの概要についても解説する．

CONTENTS

PROLOGUE

USBコントローラの種類からUSBソフトウェアの動作まで

USB機器のハードウェアとソフトウェアの構成

佐藤 陽二/桑野 雅彦

1 USBの特徴

● USBの特徴と組み込み機器におけるメリット

パソコンと周辺機器を接続することを目的としたUSB(Universal Serial Bus)の特徴は，組み込み機器においてもメリットとなります．USBの特徴をあげると次のようになります．

(1)高速/高信頼性通信

1.5Mbpsのロー・スピードから，12Mbpsのフル・スピード，480Mbpsのハイ・スピードまで対応しており，しかもデータ転送の信頼性を確保しています．

(2)プラグ＆プレイによる扱いやすさ

特に接続機器ごとの設定作業を必要とせず，電源投入状態でのケーブル挿抜ができます．また，一つのインターフェースで複数の機器と接続可能です．

(3)小型コネクタ，電力供給による省スペース，省電力化

小型コネクタのため組み込み機器における省スペース化に有効で，さらに電力供給も可能なため，小型のデバイスなら電源は不要です．

(4)パソコンとの互換性，親和性の高さ

パソコン用の安価な周辺機器やデバイスが利用可能で，標準クラス対応ならドライバを作成せずに接続が可能です．また，USBメモリなどで簡単にデータ交換ができます．

(5)ベンダ固有の拡張性が高い

ベンダ固有クラスが簡単に定義できるので応用範囲が広くなります．

(6)ホスト/デバイス構成による機能実装の容易さ

ホスト側はやや規模が大きくなりますが，その分，デバイス側は小さくなります．

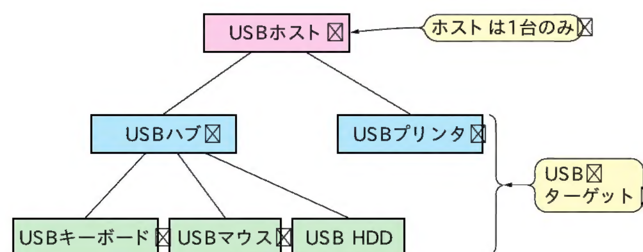


図1 USBのシステム構成

● USBのシステム構成

たとえば，EthrenetやIEEE1394などは，基本的にすべての機器が対等の立場でバスや通信路を共有しています(ハブなどの装置を除く)．よって転送を開始するにも，アービトレーション制御や衝突検出といった処理が必要になります．

USBでは通信プロトコル上，ホストとターゲット(ファンクションやペリフェラル，単にデバイスなどとも呼ばれる)が明確に分けられています．しかも，一つのUSBシステム全体の中で，ホストはただ1台しか存在できず，このホストを頂点としたツリー構造でバスが構築されます(図1)．

データ転送においても，ホストとターゲットという関係のみで転送が成立します．バス・アイドル状態から最初に転送をしようと行動を起こすことができるのは，ホストのみです．ターゲットは，ホストから自分宛ての転送データ(パケット)を受け取ったら，その内容に応じたデータをホストに返すという処理を行います．しかし，ターゲット間で直接データをやり取りすることはできません．また，ホストからの要求もないのに，ターゲットがいきなりホストにデータを送りつけることもできません．

このように，USBはホストの要求にターゲットが答えるというプロトコルのみを採用しているため，バスのスケジューリングはホストがすべて管理することになります．これにより，バスのアービトレーションや衝突検出といった処理が不要になり，ターゲットに要求されるリソースの負荷が軽くなるのです．

2 USBの活用範囲の広がり

USB規格は，比較的低価格でパソコンと周辺機器とを簡単につなぐための規格として発表され，現在は480Mbpsの高速通信に対応したUSB2.0規格に進化し，ほぼすべてのパソコンに搭載されている標準的なインターフェースとなりました．こうした流れに乗り，組み込み機器においてもUSBの活用が進んできました．

▶ 第1段階：パソコンの周辺機器

パソコン側にUSBインターフェースが搭載され，その周辺装置としてUSBインターフェースを搭載した各種組み込み機器(HID，プリンタ，マスストレージ，コミュニケーション，オーディオなど)が登場しました〔図2(a)〕．

▶ 第2段階：パソコンと同等のホスト機器

パソコン用にUSBデバイス機能を搭載した周辺機器が数多く市場に出回り，その安価かつ入手が容易な機器を利用するた

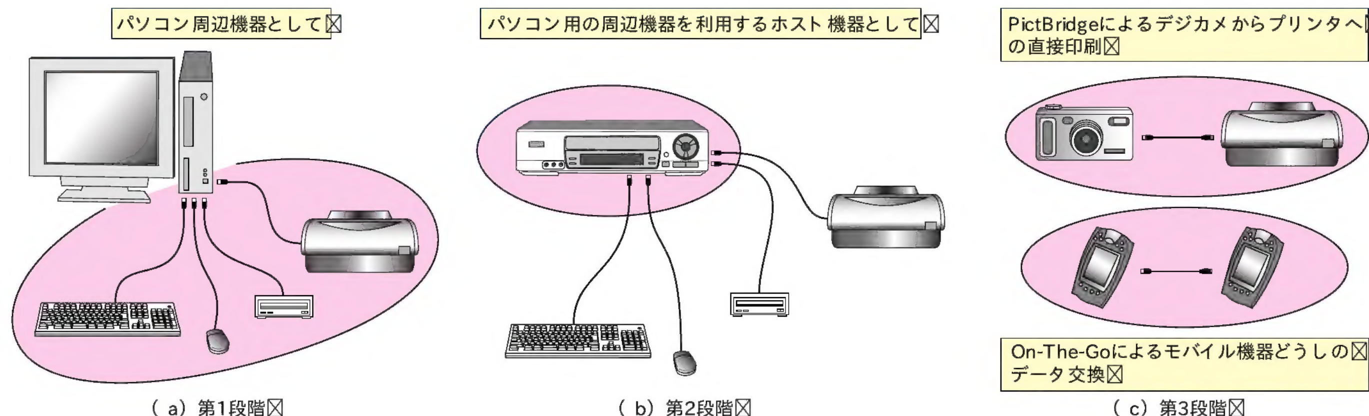


図2　USB の活用段階

め，パソコンと同等の USB ホスト機能を搭載した組み込み機器が登場しました〔図2（b）〕．

▶ 第3段階：PC レスの独自通信

これまであくまで周辺機器として存在してきた機器が，現在ではよりインテリジェントに進化してきています．周辺機器の高機能化が進むと，次のような要求も生まれます．

たとえば，PC と PDA をつなぐ場合，PC がホストで PDA がターゲットとなります．しかし，PDA どうしを接続することを考えると，ターゲットどうしになるため，接続することができません．さらに，PC とデジカメやプリンタをつなぐ場合，PC がホストでデジカメやプリンタがターゲットとなります．しかし，デジカメとプリンタを直接接続することを考えると，ターゲットどうしになり，やはり接続することができません．

これらの場合，接続しようとするどちらかの機器がホストの役割を演じる必要があります〔図2（c）〕．

そこで登場してきたのが，On-The-Go や PictBridge などの仕様です．第2段階までは，あくまでもパソコン向けに用意された通信規格を PC や組み込み機器へ応用するにとどまっていましたが，On-The-Go や PictBridge などの登場により，これまで周辺機器としてしか接続できなかった機器どうしを接続できるようになりました．今後さらにディジタル家電やモバイル機器などへの応用が進むことが期待されます．

3 USB 機器の基本構成

● USB 機器のハードウェア構成

USB 機器を実現するには，一般的には USB コントローラを採用し，ソフトウェアと組み合わせて USB 機器を開発します．

当然ながら USB コントローラには，ホスト用の USB コントローラ，ターゲット用の USB コントローラ，そしてどちらの機能も内蔵したホスト/ターゲット両用の USB コントローラがあります．

（1）USB ターゲット・コントローラ

周辺機器としてだけ動作する機器であれば，採用するコントローラはターゲット専用のこのタイプで十分でしょう．このタイプのコントローラには，さらに次のような形態のものがあります．

● ワンチップ・マイコン USB コントローラ

もっとも規模の小さい USB 機器なら，メモリも内蔵したワンチップ・マイコンといっしょになった USB コントローラがあります．これ一つで USB 機器を実現できます．

キーボードやマウスなど，ロー・スピード対応のもっともコスト重視のコントローラは，このタイプが多いようです．

● CPU 内蔵 USB コントローラ（メモリは外付け）

ワンチップ・マイコンではメモリ容量が少ないという場合は，メモリは外付けするが，CPU 内蔵周辺機能の一つとして USB ターゲット・コントローラが内蔵されている CPU もあります．

本特集では，第2章で取り上げている SH や H8 内蔵 USB コントローラがこのタイプです（一部メモリ内蔵タイプもある）．

● USB コントローラ単体

CPU は内蔵していないので，何らかの CPU のローカル・バスに接続して使います．CPU の種類やメモリ容量なども自由に選べるので，もっとも自由度の高いシステムを設計できます．

COLUMN

USB2.0 ≠ハイ・スピード

USB2.0仕様といえばハイ・スピード対応で，USB1.1仕様がフル/ロー・スピード対応と思っている人も多いようですが，そこにはやや誤解があります．

現在 USB の最新仕様は USB2.0であり，フル/ロー・スピードもすべて包含したものとなっています．いまでも便宜的にハイ・スピードを USB2.0，フル/ロー・スピードを USB1.1と呼ぶこともありますが，正しい表現としては USB2.0ハイ・スピードであり，USB2.0フル/ロー・スピードなのです．よって USB2.0仕様であってもハイ・スピードに対応していない場合もあります．

本特集では，第1章で取り上げているISP1582が，このタイプです．

ターゲット・コントローラには標準仕様というものはなく，各デバイス・ベンダがそれぞれのコアをシリーズ展開しています．同じファミリであれば，上位互換になっているものが多いので，ファームウェアを共通にするといった設計も可能です．

（2）USBホスト・コントローラ

PCや，つねにホストとして動作する組み込み機器で採用するコントローラです．このタイプにもいくつか種類があります．

ホスト・コントローラには標準仕様がいくつか存在します．Intel製チップセットに内蔵されているUHCI，MicrosoftやPCベンダ主導で規格化されたOHCI，そしてハイ・スピード対応のEHCIがあります．

●PCIバス・ベースのフル仕様タイプ

PC向けチップセットやPCIバスに接続するタイプのUSBホスト・コントローラです．一般的に，UHCI/OHCI/EHCIのいずれかの仕様に準拠しています．

●高機能組み込み向けCPU内蔵タイプ

情報家電機器向けの高機能な32ビットRISCマイコンなどに，周辺機能の一つとしてUSBホスト・コントローラを内蔵したものがあります．コントローラ仕様としては，OHCIに準拠したものが多いようです．

●組み込み向け簡易タイプ/高機能タイプ

このタイプは，実際にはホストとターゲットの両方の機能を内蔵したものが多いようです．

第3章で取り上げているSL811は簡易タイプ，第4章で取り上げているM66596は高機能タイプといえるでしょうか．

（3）On-The-Go対応コントローラ

使用する場面により，ホストにもターゲットにもなりうる機器であれば，このタイプのコントローラを採用します．

第6章で取り上げているML60842や，第7章で取り上げているISP1362がこのタイプです．

前述のホストとターゲットの両機能を内蔵したタイプとの違いは，On-The-Go機能の有無です．前述のタイプのコントローラを採用する場合，機器にターゲット用のコネクタ（タイプB）とホスト用のコネクタ（タイプA）の両方を実装するのが普通です．On-The-Go対応コントローラの場合は，一つのコネクタでホストにもターゲットにも対応できるMini-ABと呼ばれるコネクタを採用できます．



図3 USBソフトウェアの基本構成

● USB機器のソフトウェア構成

ホストであれターゲットであれ，USBのソフトウェアは基本的に図3のような階層構造をとります．

（1）USBコントローラ・ドライバ

USBコントローラに依存したドライバ層です．コントローラに対するI/O処理，割り込み処理，コントローラに依存するスケジューリング処理などを行います．

（2）USBドライバ

コントローラやクラスに依存しないUSBの論理的な通信制御処理を受け持つドライバ層です．また，接続されたUSBデバイスの構成を管理する機能も一般的にこの層が管理します．

（3）クラス・ドライバ

接続するUSB機器のクラスに対応したエンドポイントの管理や通信プロトコル処理を行うドライバ層です．

（4）上位プロトコル

USB転送上にマッピングされた上位のプロトコル層です．USB側から見ればアプリケーションとみなすことができるものですが，USBの応用システムを開発する立場からすれば純粋なアプリケーション・ソフトウェアとは区別することになるので，あえて別の階層として記述しました．

（5）アプリケーション

USB機器を利用したアプリケーション・ソフトウェア層です．

4 WindowsにおけるUSBシステムの構成

● WindowsにおけるUSBハードウェアの構成

Windowsの場合，そのほとんどがPCIバス・ベースのUHCI/OHCI/EHCIのホスト・コントローラを搭載しています．また，Windowsマシン自体をUSB周辺機器として使うことはないため，ターゲットとしての機能はありません（PCのUSBポートをつないでファイル転送を実現するような接続ケーブルは，ケーブル自体がUSBターゲットとして動作している）．

● WindowsにおけるUSBソフトウェアの構成

WindowsにおけるUSBソフトウェアの構成を図4に示します．

Windowsでは，プラグ&プレイ機能により，デバイスが接続されたときに必要なモジュールを検索して動的にロードするようなしくみを持っています．そのため，初めて接続した機器で，まだドライバがインストールされていない場合でも，自動的にドライバ・インストール用のダイアログが表示され，そこで必要なドライバを指定しインストールすることで，初めて接続した機器もそのまま利用できるようになります．また，2回目以降は自動的にそのドライバがロードされるようになり，特に設定は必要ありません．

● Windowsにおける「USBドライバの作成」とは

図4でわかるように，UHCI/OHCI/EHCIの各ホスト・コン

Information —— Windows CE初のバックドア型トロイの木馬が発見される
（株）シマンテックによると，Windows CEのバックドア型トロイの木馬「Backdoor.Brador.A」が発見された．バックドアの影響を受けるのは，ARMコアCPU搭載のPocketPCのみ．

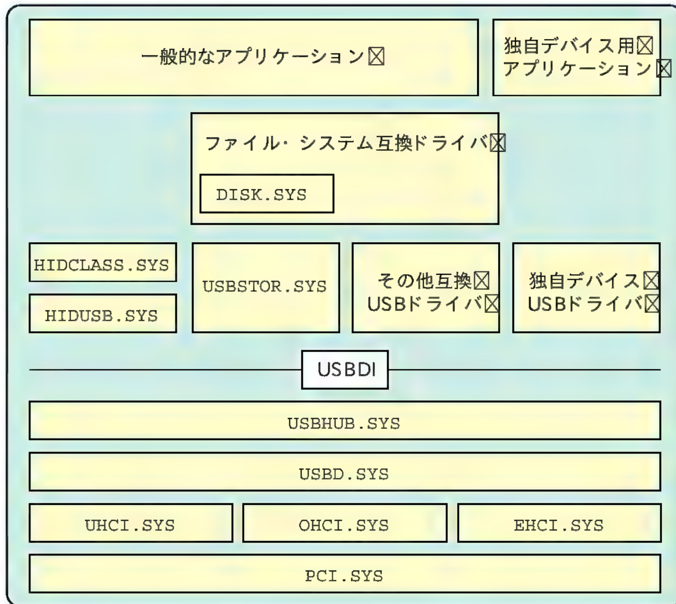


図4 Windows における USB ソフトウェア構成

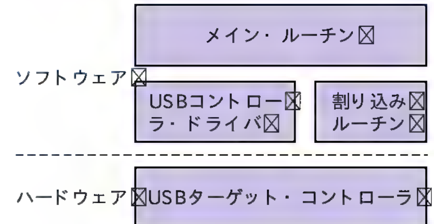


図5 もっとも簡単な USB ターゲットのソフトウェア構成例

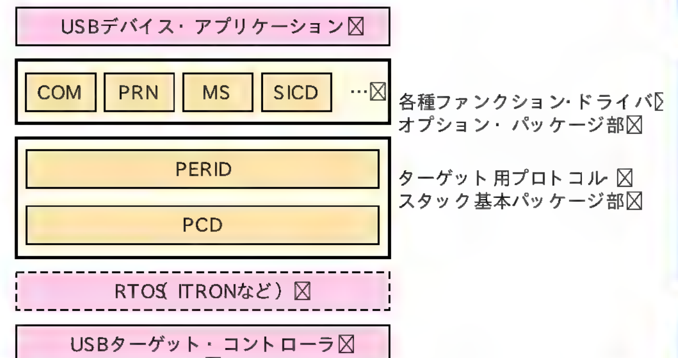


図6 開発効率/移植性を考慮した USB ターゲットのソフトウェア構成

トローラのドライバや USB ドライバ本体，USB ハブ用ドライバなどは，すでに Microsoft 社から OS 標準ドライバという形で提供されています．

よって，Windows 環境で「USB ドライバを作成する」といった場合は，図4にある「独自デバイス用 USB ドライバ」から上位階層部分を作成することを意味しています．

● Linux や BSD などの場合

Linux や BSD など規模の大きな OS も，基本的には Windows と同様な階層構造を取っています．これらの OS 上で USB ドライバを開発する場合も，ある部分から上の部分のみを作成すれば良いようになっています．

5 組み込み機器における USB システムの構成

● 組み込み機器における USB ハードウェアの構成

3節で説明したように，規模や利用形態に合わせて，それに最適な USB コントローラを選択する必要があります．

● 組み込み機器における USB ソフトウェアの構成

組み込み機器はパソコンに比べてハードウェア資源の制約が非常に厳しいものになっています．また，組み込み機器は用途がある程度限定されているため，接続対象とする機器も限られたものとなります．そうしたことから，組み込み機器向けの USB ソフトウェアは，必要最小限の機能をあらかじめ組み込んで静的な構成にするのが一般的です．

▶USB ターゲットの場合

もっとも簡単な USB ターゲットのソフトウェア構成例を図5に示します．USB コントローラに直接アクセスするルーチンと，割り込みルーチン，そしてメイン・ルーチンからなる簡単なもので，RTOS もありません．規模も小さく，実験/試験的な場合や趣味的な開発であれば，これでも十分でしょう．しかし，メイン・ルーチン部分をほかの USB コントローラに移植しようとすると，USB コントローラの構造の違いからくる仕様変更などで，たいへんな手間がかかることもあるでしょう．

開発効率や移植性を考慮した，より本格的な USB ターゲットのソフトウェア構成例を図6に示します．規模が大きくなれば，USB の処理だけでなく，それ以外の処理も増えてくるので，RTOS によるタスク切り替えなどは必須となってくるでしょう．

USB の場合，ホスト側に比べデバイス側の処理がそれほど多くなく，使用するコントローラによってはほとんどソフトウェアが介在する必要がない場合もあります．ただし，最近の USB デバイス機器では複数の機能を搭載したり，USB の上位アプリケーションの規模が大きくなる傾向があります．そうしたことから上位のソフトウェアの再利用性を高めるために汎用の USB デバイス・ドライバを利用するケースが増えてきています．

▶USB ホストの場合

組み込み機器における USB ホストのソフトウェア構成例を図7に示します．

基本的には Windows と類似したソフトウェアの階層構造をとりますが，前述のとおり，あらかじめ必要なモジュールを組み込んだ静的な構造となり，サポート対象外の機器は接続できません．

▶On-The-Go の場合

On-The-Go 対応の場合のソフトウェア構成例を図8に示しま

す．On-The-Goの場合，前述のホストおよびターゲット両方のスタックのほか，On-The-Go規格に準拠したSRP，HNPプロトコル処理およびホスト/デバイスの切り替え処理を行うためのOTGスタックが搭載されます．

● 組み込み機器における「USBドライバの作成」とは

組み込み機器でも，規模の大きなOSを採用した場合は，OSにあらかじめUSBのドライバが用意されています．この場合はWindowsなどと同様に，ユーザは独自のクラス・ドライバやその上位アプリケーション部分を作成することになります．

ITRONなど比較的軽い仕様のOSを採用した場合は，USB関連ドライバは含まれていない場合が多いので，ミドルウェア・メーカなどから販売されているUSB用プロトコル・スタックを購入し，その上に独自クラスのドライバやアプリケーションを作成します．仕様の軽いUSBターゲット機器では，足回りのドライバからすべてを自社開発する場合もあるでしょう．

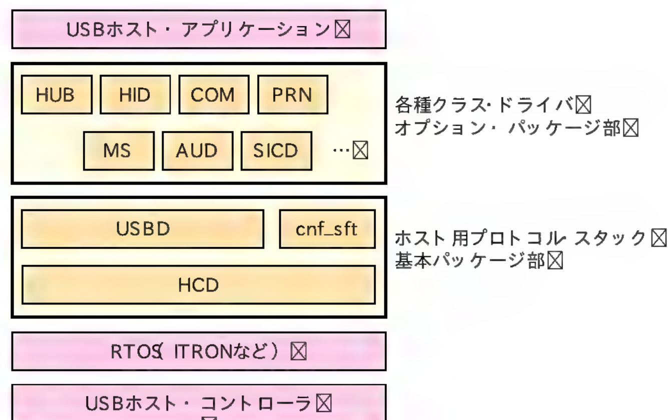


図7　組み込み機器による USB ホスト のソフトウェア構成例

6 各種用途におけるソフトウェア構成例

● キーボード/マウス（HIDクラス）

キーボード/マウスなどのHIDを使用するUSBホストの構成例を図9に示します．

キーボード/マウスを使用するためにはHIDクラス・ドライバが必要です．HIDクラス・ドライバでキーボード/マウスに関してどこまで処理しているのかは，ミドルウェア・ベンダにより実装が異なるので個別に確認が必要です．

また，キーボードとマウスを同時に接続して使用する場合にはハブが必要になり，一般にハブ・クラス・ドライバが必要となります．なお，USBコントローラによっては，複数のポートをサポートしており，そのポート数まではハブが必要ないものもあります．

● シリアル通信（Communicationクラス）

USBでシリアル通信をエミュレートするようなシステム構成例を図10に示します．

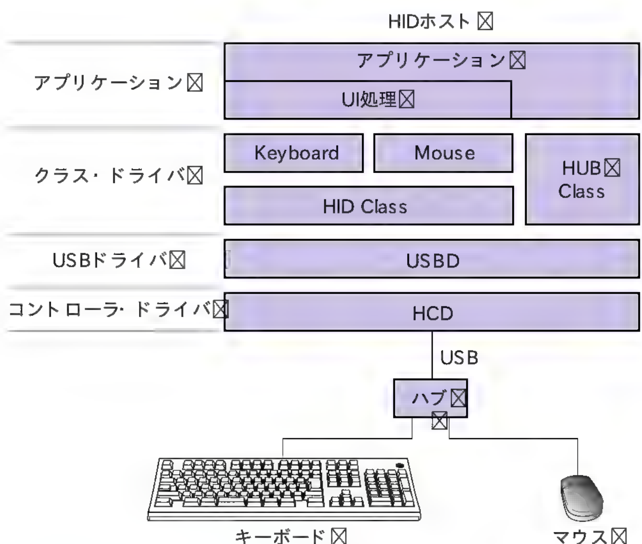


図9　HIDを使用する USB ホスト の構成例

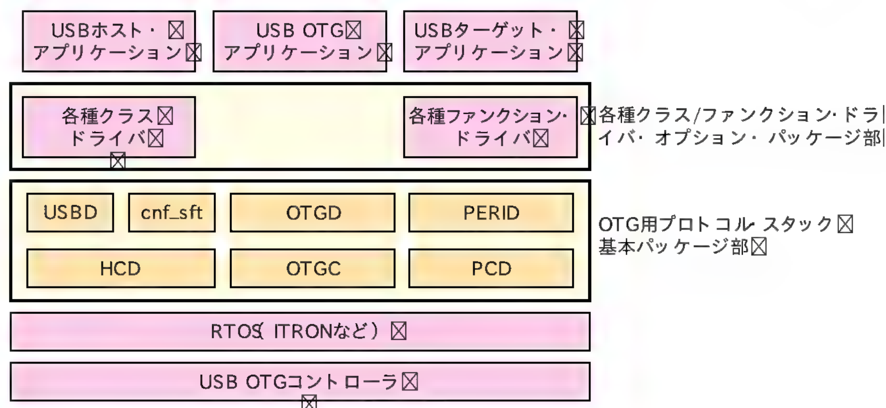


図8　On-The-Goのソフトウェア構成例

COLUMN

USBプロトコル・スタック

組み込み機器向けのUSBホスト・コントローラ・ドライバや各種クラス・ドライバは，USBプロトコル・スタックとして各社から販売されています．

実は図6～図14のソフトウェア構成例は，(株)グレープシステムより発売されているGR-USBシリーズのものです．他社のUSBプロトコル・スタックでも，基本的な考え方は同様と思われます．

(株)グレープシステム
http://www.grape.co.jp/

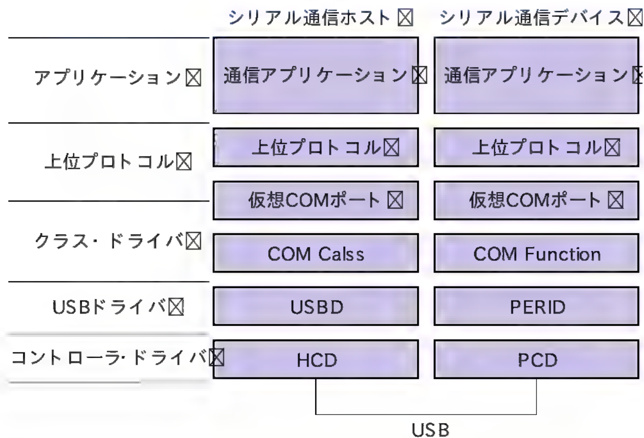


図10　シリアル通信をエミュレートするシステムの構成例

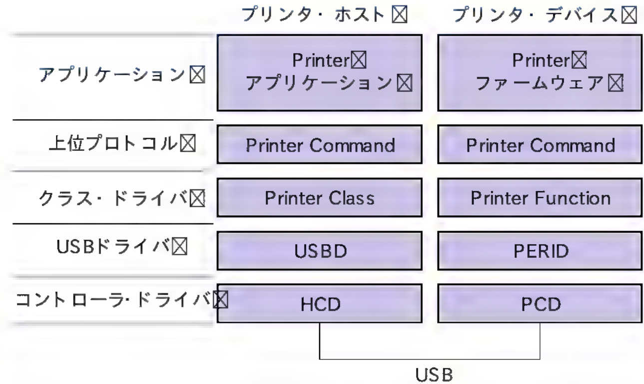


図11　プリンタ・ホスト/デバイスの構成例

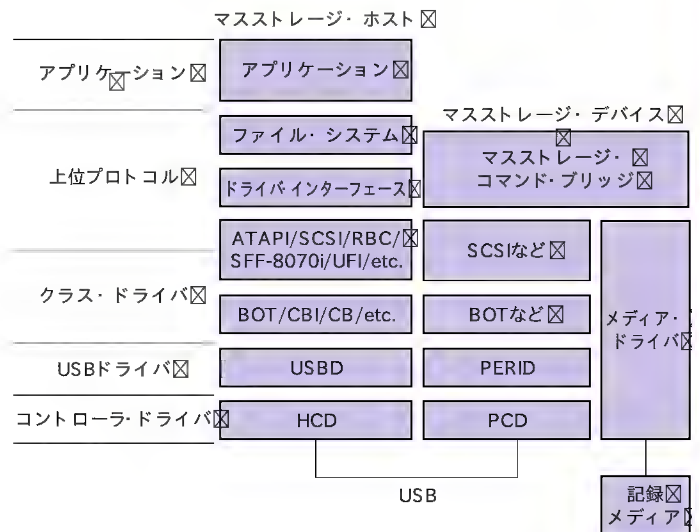


図12　USBを利用したファイル・システムの構成例

一般にホスト，デバイス側ともコミュニケーション・クラスの上位に仮想COMポートのようなAPIが提供されます．そのため，上位の通信プロトコルやアプリケーションは，基本的にシリアル通信と同等プログラミング・モデルを使用でき，それほどUSBを意識することなく実装が可能です．

● プリンタ（Printer クラス）

USB接続を使用したプリンタ・ホスト/デバイスの構成例を図11に示します．

USBではプリンタ・クラスが定義されていますが，このクラスは各プリンタ固有のコマンドに関して一切規定していないので，別途上位プロトコルとして各プリンタ固有のコマンド処理を実装する必要があります．

● ファイル・システム（MassStorage クラス）

ファイル・システムでUSBのマスストレージ・デバイスを使用するホストおよびマスストレージ・デバイスのソフトウェア構成例を図12に示します．

ホスト機器側では，ファイル・システムの下にマスストレージ・クラス・ドライバが必要です．マスストレージ・クラスには転送プロトコル種別としてBOT，CBI，CBなど，その上位にATAPI，SCSI，UFI，SFF-8070iなど各種コマンド体系に対応したサブクラスがあり，ホスト側では接続するデバイスがサポートするプロトコル，コマンド体系にあわせたサブクラスを実装する必要があります．

なお，USBのマスストレージ・クラスは各種コマンドを受け渡すための機能を提供しますが，そのコマンドをどのように使用するかについて規定しているものではありません．さらに，ファイル・システムとUSBスタックは別々のプロダクトとして提供されることが多く，それらは直接インターフェースされていません．そうしたことから，ファイル・システムの下位ドライバとしてUSBのマスストレージ・クラスの機能を使用して，適切なコマンドを発行するドライバ・モジュールを用意する必要があります．

マスストレージ・デバイス側は，USBのマスストレージ・クラスのコマンドを，そのままメディア・ドライバのコマンドに変換して受け渡す程度の比較的簡単な処理の実装で実現可能です．

● ネットワーク（ベンダ固有クラス）

USBネットワーク・アダプタを利用したシステム構成例を図13に示します．

ホスト側ではTCP/IPなどのネットワーク・プロトコルの下に，各ネットワーク・アダプタ用のクラス・ドライバが必要です．現在，EthernetやWireless LANなどの各種ネットワーク・アダプタが市場に出回っていますが，ほとんどの機種はベンダ固有のクラスが使用されおり，それぞれ固有のクラス・ドライバが必要となります．また，TCP/IPなどのネットワーク・プロトコルによって下位のドライバ・インターフェースが異なるため，それぞれにあわせたドライバ・インターフェース・モジュールが必要になります．

TCP/IPなどのネットワーク・プロトコルより上位のプロトコルやアプリケーションは特にUSBを意識することはありません．

● PictBridge（SICD クラス）

PictBridgeのシステム構成例を図14に示します．

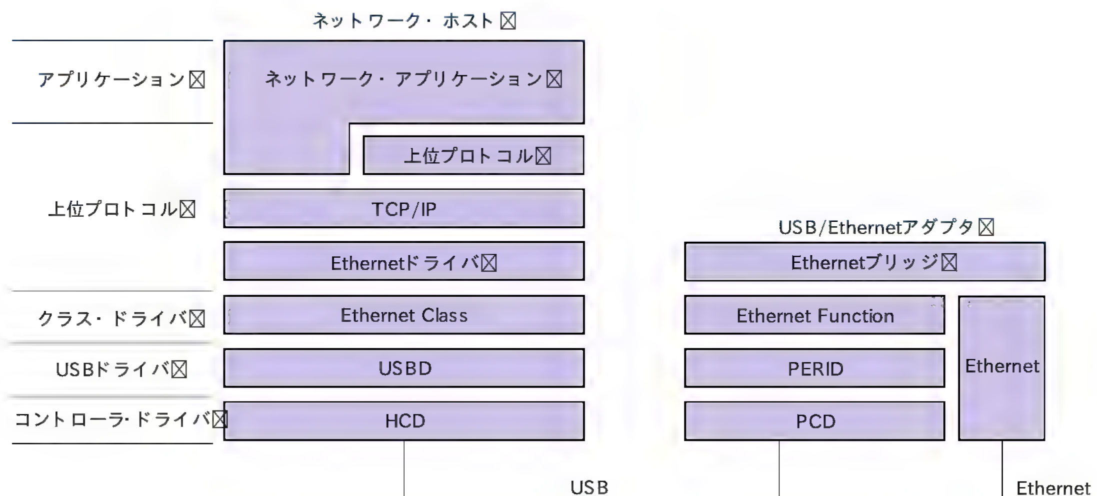


図13　USBネットワーク・アダプタを使用するシステム構成例

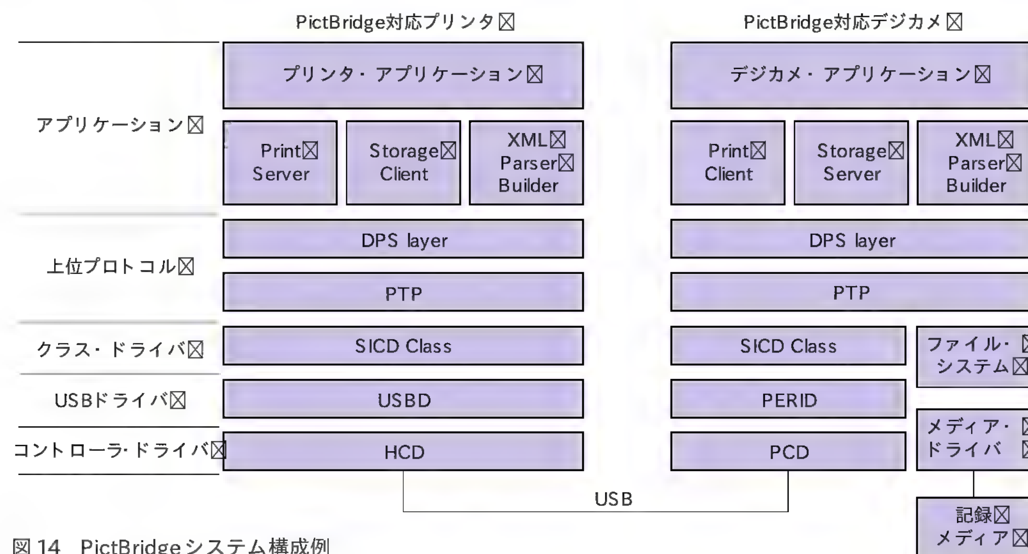


図14　PictBridgeシステム構成例

PictBridgeでは，デジカメなどの画像入力デバイス側がUSBデバイス，プリンタなどの画像出力デバイス側がUSBホストとなります．画像データや印刷制御コマンドをPTPプロトコルで受け渡すことによりPCレスの画像印刷処理を実現しています．

PictBridgeに関する説明はAppendix2で解説します．

7　組み込み機器における USBソフトウェアの動作概要

組み込み機器におけるUSBソフトウェアの処理フローの全体を図15に示します．

● 初期化

(1) ドライバの初期化

各USBドライバ・モジュールの管理情報を初期化し，必要に応じて内部タスク，同期オブジェクトなどの生成および初期化を行います．

(2) コントローラの初期化

USBコントローラおよび関連ハードウェアの各種レジスタを初期化します．

(3) 割り込みの登録

USBコントローラおよび関連ハードウェアからの割り込み処理を登録します．

(4) コールバック・ルーチンの登録

上位アプリケーションに対する各種イベント通知用のコールバック・ルーチンなどを登録します．

(5) コントローラの起動

各種初期化が終了した後，USBコントローラおよび関連ハードウェアを動作可能な状態とします．

● 接続

(1) デバイス接続の認識

USBコントローラからの接続割り込み発生，あるいはハブからの接続通知により，ホスト側でデバイスの接続が認識され

ます．

（2）バス・エニュメレーション処理

接続されたデバイスに対するバス・エニュメレーション処理が起動され，各種ディスクリプタの取得，デバイス・アドレスの設定のほか，必要に応じてコンフィグレーションの設定などが行われます．

（3）パイプのオープン処理

取得したデバイス情報から該当するクラス・ドライバの接続処理が呼び出され，そのクラスのデータ転送に必要なパイプのオープン処理などが行われます．

（4）上位アプリケーションへの接続通知

データ転送の準備が整った後，上位アプリケーションに対して接続通知のコールバックが行われます．上位アプリケーションでは接続されたデバイスと通信を開始する準備を行います．

● データ転送

デバイスとの転送用にオープンしたパイプを使用して各種データ転送（標準/ベンダ固有デバイス・リクエスト，バルク，インタラプト，アイソクロナス転送）を行います．

通常はクラス・ドライバが提供するAPIを使用してデータの送受信を行うため，直接パイプに対する転送要求を発行する必要はありません．

転送要求の完了は，コールバックなどで通知される非同期型と，転送要求APIが完了まで待ち状態となる同期型がありますが，組み込み向けのUSBスタックではコードのサイズや必要資源を少なく抑えるため非同期型としているものが多いようです．

データ転送の要求方法は，USBソフトウェアの構成によって異なります．基本的に次のような三つのケースがあります．

（1）USBドライバ直接

クラス・ドライバが用意されていない場合や，独自にデバイス・リクエストを行うような場合には，USBドライバのAPIを直接コールします．

USBドライバの一般的なAPIとしては，各パイプに対する通常の転送要求（バルク/インタラプト/アイソクロナス）や，各種デバイス・リクエスト転送要求（コントロール）などが用意されており，そのAPI仕様は各ドライバにより異なります．

（2）クラス・ドライバ経由

クラス・ドライバまで提供されており，その上位アプリケーションを開発するような場合は，クラス・ドライバのAPIを使用してUSBデータ転送を行います．

クラス・ドライバのAPIは一般的にそのクラスの提供する機能ごとに提供されます．たとえば，マスストレージ・クラスの場合，INQUIRY，READ（10），WRITE（10），SEEK，TEST UNIT READYなどの各コマンドに対応したAPIが提供されます．

なお，クラス・ドライバのAPIが，どの程度USBのデータ転送を意識するのかについては，それぞれのクラス・ドライバの実装により異なります．

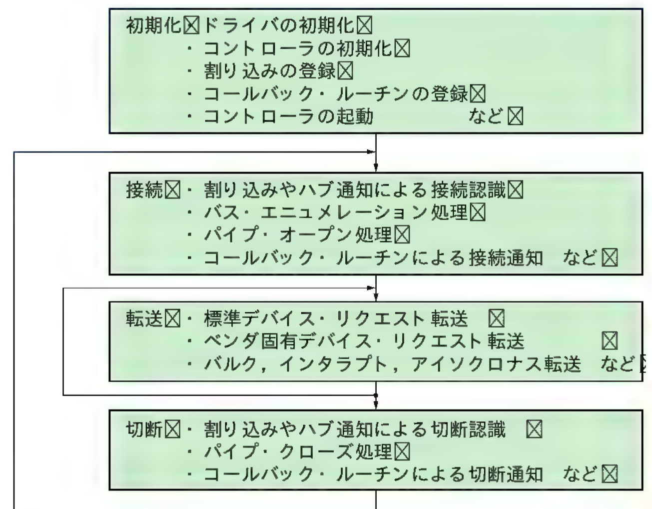


図15　USBソフトウェアの全体処理フロー

（3）上位ミドルウェア経由

USBの上位にミドルウェアが搭載されている構成の場合には，そのミドルウェアのAPIをコールすることによりUSBデータ転送が実行されます．これらのAPIは基本的にUSBのデータ転送を意識しません．

このような例としては，USBシリアル変換ケーブルなどの仮想COMポート，USBマスストレージ・デバイスをサポートしたファイル・システム，USB-LANアダプタをサポートしたネットワーク・プロトコルなどがあります．

● 切断

（1）デバイス切断の認識

USBコントローラからの切断割り込み発生，あるいはハブからの切断通知により，ホスト側でデバイスの切断が認識されます．

（2）パイプのクローズ処理

切断されたデバイスとのデータ転送用パイプをクローズ状態にします．

（3）上位アプリケーションへの切断通知

上位アプリケーションに対してコールバックなどで切断が通知されます．上位アプリケーションでは，現在処理中の転送要求の中断処理およびそれに伴う後処理などを行います．

まとめ

おおまかですが，組み込み機器における一般的なUSBシステム構成と，USBのソフトウェアの動作概要を解説しました．

本稿が，これからUSBインターフェースを搭載した組み込み機器を開発しようとする人にとって，USBソフトウェア開発はどのようなものなのかをイメージするための一助となれば幸いです．

さとう・ようじ　(株)グレープシステム
くわの・まさひこ　パステルマジック

1

ハイ・スピード対応汎用USBターゲット・コントローラ

ISP1582を使ったUSB機器の開発事例

東山 謙

第1部では，USBターゲット・コントローラについて解説する．ここでは480Mbpsのハイ・スピード転送にも対応したUSBターゲット・コントローラISP1582について解説する．OTG（On-The-Go）ペリフェラル・デバイスとしても使えるコントローラなので，携帯機器などに最適なUSBターゲット・コントローラである． （編集部）

はじめに

ISP1582は，USB規格の策定メンバであるオランダのPhilips Semiconductors社（フィリップス）製のもっとも新しいハイ・スピードUSB2.0デバイス・コントローラであり，480Mbpsのデータ・レートをサポートしています．またOTG規格1.0aにも対応しており，OTGペリフェラル（第6章のコラムを参照）としても動作が可能です．今回はISP1582 PCI評価キットを使用して，USB機器の開発方法について説明します．

1 ISP1582の機能

ISP1582は，図1に示すようにUSB2.0トランシーバ，プロトコル・エンジン（SIE/PIE），エンドポイント用RAM（8Kバイト），CPUインターフェース，DMAインターフェースなどを備えた，ワンチップ汎用デバイス・コントローラです．システムCPUからISP1582を制御するので，ほぼすべてのデバイス・クラスに対応が可能です．また，インターフェース電圧も1.65V～3.6Vの範囲で動作可能なので，低電圧動作の携帯機器でも使用することができます．

パッケージは写真1に示すよう，8mm角の56ピン小型パッケージ（HVQFN56）を採用しています．

● USB2.0トランシーバ

ISP1582はUSB信号（D+/D−）を直接入出力するアナログ・トランシーバです．USB規格に従い，ハイ・スピードとフル・スピードの両方のトランシーバを内蔵しています．接続先

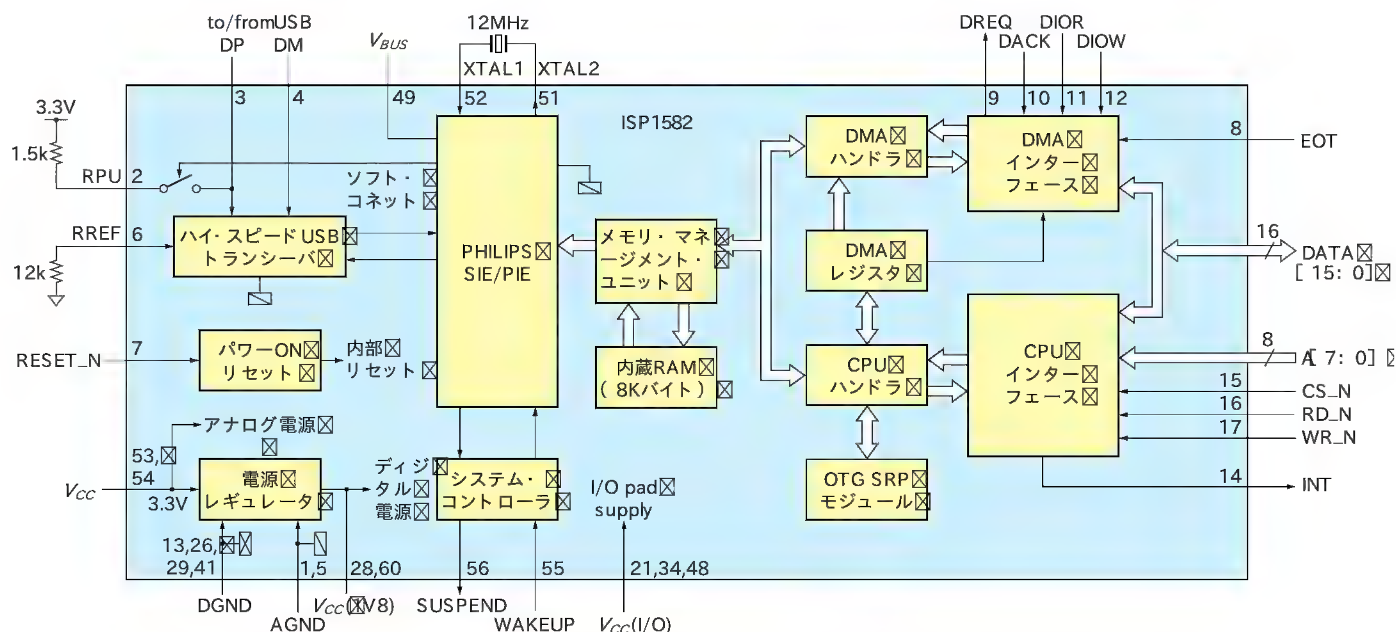


図1 ISP1582のブロック図


写真1 ISP1582パッケージ外観

表1 CPU/DMAインターフェース信号

DATA[15:0]	データ・バス
A[7:0]	アドレス・バス
CS_N	チップ・セレクト
RD_N	リード
WR_N	ライト
INT_N	割り込み

(a) CPUインターフェース

DATA[15:0]	データ・バス
DREQ	DMAリクエスト
DACK	DMAアクノリッジ
DIOR	DMAリード
DIOW	DMAライト
EOT(オプション)	転送終了

(b) DMAインターフェース

のホストやハブがサポートしているスピードを検出して，どちらのトランシーバを使用するかを自動的に決定し，切り替えます．

● プロトコル・エンジン(SIE/PIE)

次に示すプロトコル・レイヤをハード・ワイヤードで実装しています．

- Syncパターン認識
- パラレル－シリアル変換
- ビット・スタッフィング/デスタッフィング
- CRCのチェック/生成
- PIDのチェック/生成
- アドレス認識
- ハンドシェークのチェック/生成

USB規格の第8章で規定されている機能を，完全にサポートしているため，これらに対するファームウェアの介在は必要ありません．

● エンドポイント構成

ISP1582では，64バイトのコントロール・エンドポイント以外に，最大で7本のインと7本のアウト・エンドポイントを同時に使用することが可能です．コントロール，インタラプト，バルク，アイソクロナスのすべての転送モードをサポートしています．

エンドポイント用のRAMは，全体で8Kバイトを内蔵しており，これをそれぞれのエンドポイントに自由に割り当てることが可能です．またダブル・バッファ設定を行うことも可能です．

● CPU/DMAインターフェース

データ16ビット，アドレス8ビットの汎用インターフェースを持っています．アクセス・サイクルは50nsの能力があります．表1(a)に示すCPUインターフェース信号で制御を行います．

また，USB2.0規格の特徴である，高速転送を実現するために，ISP1582はDMAをサポートしています．CPUインターフェースと同じデータ・バスを使用してDMA転送を行います．表1(b)にDMAインターフェースの制御信号を示します．

DMA Configurationレジスタの設定により，DMAのバス幅を8ビットにするか，16ビットにするかの選択が可能です．また，DMA Hardwareレジスタを設定することにより，DREQ，DACK，EOT，DIOW，DIOR信号の極性を変更することができます．また，DMAのバス幅を16ビットで使用する場合には，エンディアンの変更も同レジスタにより設定できます．

● インタラプト・メカニズム

ISP1582がアサートする割り込み信号の極性とタイプ(エッジまたはレベル)は，Interrupt Configurationレジスタで設定を行います．図2にインタラプト・メカニズムを示します．

ISP1582からの割り込みの許可は，ModeレジスタのGLINTENAビットにより制御が可能です．各割り込み要因は，Interruptレジスタにレポートされます．割り込み要因としては，各エンドポイントの転送終了のほかに，Bus Reset，SOF，疑似SOF，Suspend，Resume，DMA，Vbus，High-Speed Status Changeがあります．アサートされた要因は，Interrupt Enableレジスタの対応するビットがセットされていれば，割り込み信号として外部のCPUにレポートされます．DMAを使用する際には，DMA Interrupt Reasonレジスタに，外部EOT，内部EOT，転送終了の要因がセットされ，DMA Interrupt Enableレジスタの対応するビットがセットされていれば，InterruptレジスタのDMAビットがセットされるしくみになっています．

2 ISP1582 PCI評価キットの概要

● PCIバス対応評価キット

ISP1582 PCI評価キットは，写真2のようにPCIインターフェースを備えた評価ボードで，PCのPCIスロットに挿して動作します．ボード上にCPUは搭載されておらず，PCのプロセッサからPCIバスを経由して，ISP1582の制御を行う構成に

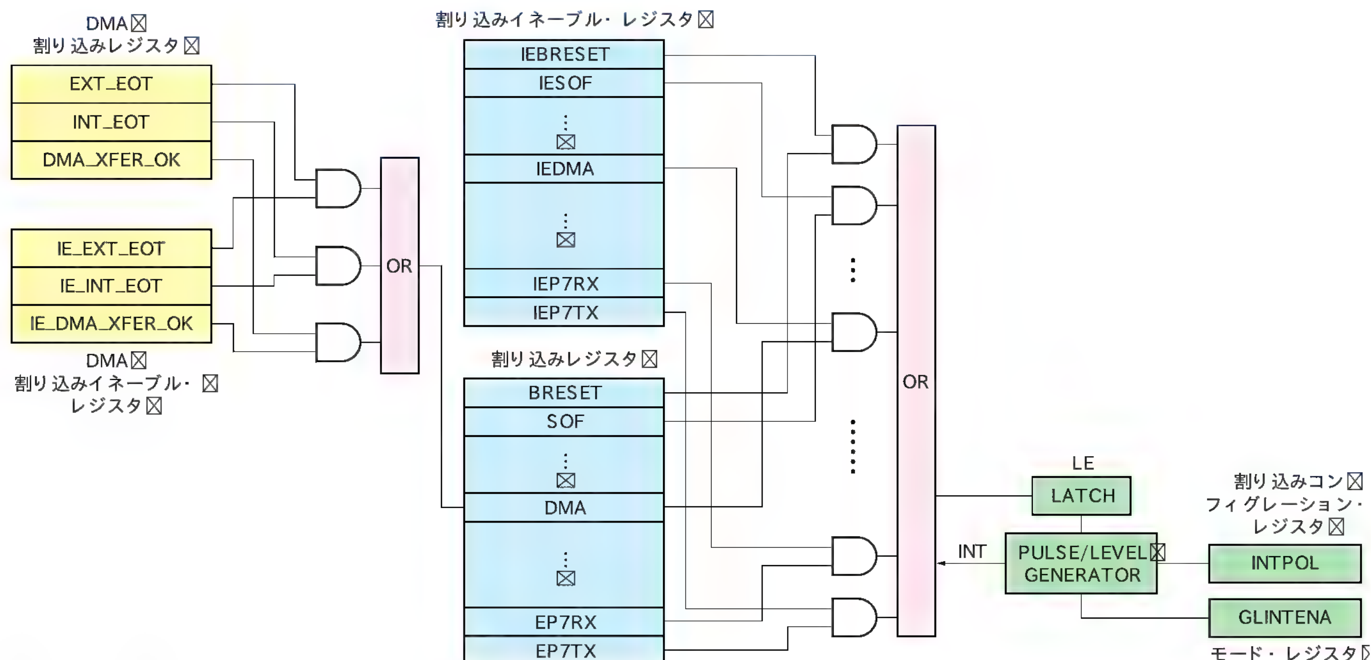


図2　ISP1582のインタラプト・メカニズム

なっています．評価用のファームウェアは，Turbo-C3.0で開発されており，DOSベースで動作するので，Windows98やWindows Meなどの16ビット系のOSが必要になります．評価キットとPCで，USBデバイスの機能を実現するということになります．USBホスト側のPCは，USB2.0搭載のWindows 2000またはWindows XPマシンを使用します．

図3に評価キットのブロック・ダイヤグラムを示します．

● バス・トランスレータFPGA

USBケーブルからのD+/D−信号の送受信を行います．CPU/DMAインターフェースはPCIバスに直接接続することはできないので，インターフェース信号はすべて，FPGA（Spartan XCS30XL，ザイリンクス）に接続されています．

FPGAは，ISP1582のCPU/DMAバスとPCIブリッジPCI9054（PLXテクノロジ）のローカル・バス間の信号とタイミングを調整するように動作します．また，DMAコントローラとしても機能します．電源ON時に，FPGA用シリアルPROM（XCS17S30XL，ザイリンクス）からプログラムがロードされます．

● PCIブリッジ

PCI9054はPCIブリッジとして動作し，PCIとローカル・バスの変換を行います．ローカル・バスは，PCI9054のCモードと呼ばれる，アドレス・バスとデータ・バスがマルチプレクスされていない通常のバス・モードを使用しています．また，PCIバスの割り込みはINTAを使うようになっています．これらの設定は，電源ON時にPLX用シリアルPROM（AT93C86，アトメル）からロードされます．

● インターフェース信号ヘッダ

ISP1582のすべてのインターフェース信号にはヘッダが取り付けられているので，これを利用して，波形の観測を行うことが可能です．また，FPGA用シリアルPROMを外せば，FPGAのインターフェース・ピンがハイ・インピーダンス状態になるので，ヘッダから信号線を引き出し，別のCPUボードに接続してファームウェアの開発を行うことも可能です．

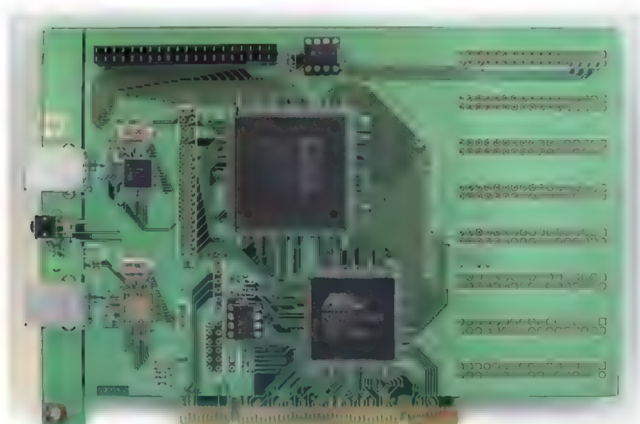


写真2　ISP1582 PCI評価キットの外観

3 PCI評価キットの動作確認

● セットアップ方法

ISP1582 PCI評価キットには，評価ボードを動作させるためのファームウェア（Pcikit.exe）とホスト側で使用するドライバ（Phkit.sys），そして評価用アプリケーション（UsvDevice.exe）が同梱されています．これらを使用して，PCI評価ボード

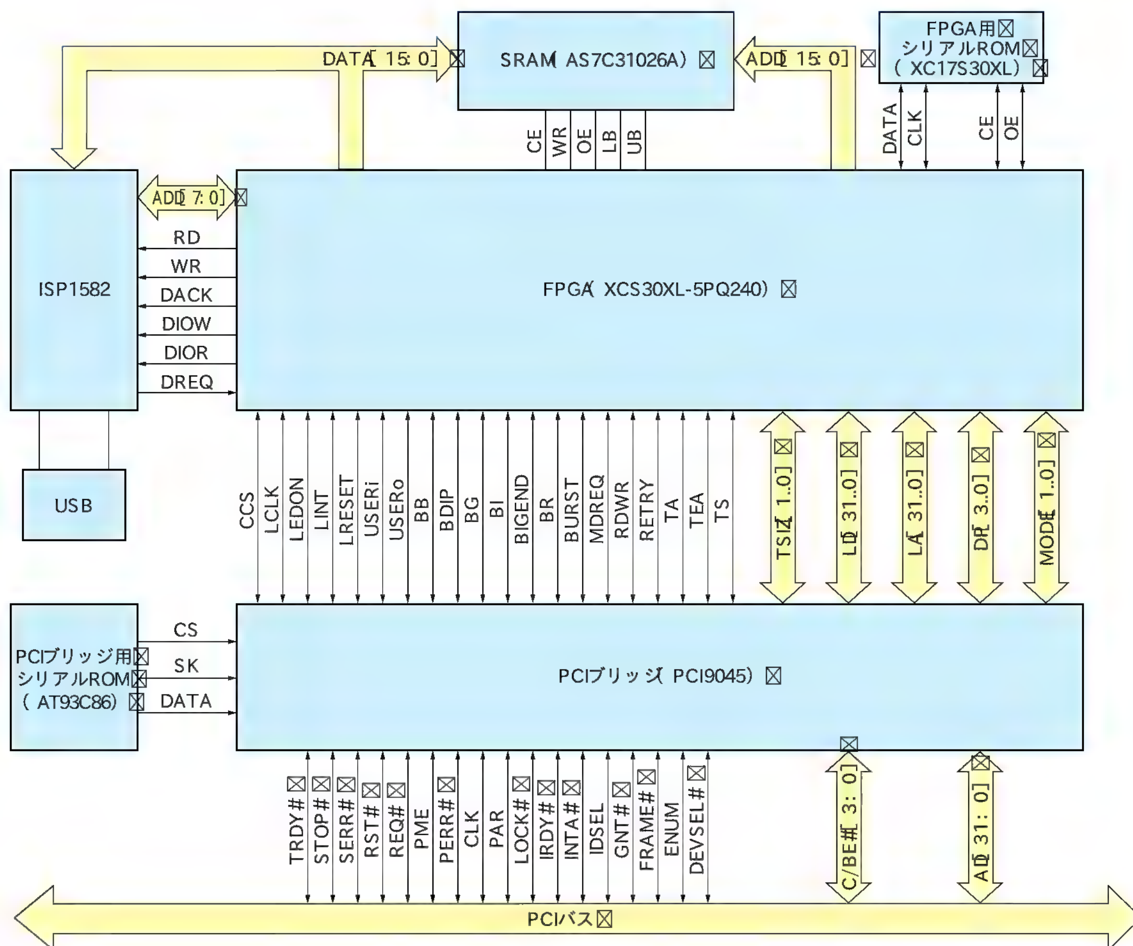


図3　ISP1582 PCI評価キットのブロック・ダイヤグラム

の動作を確認することができます．

評価ボードにはジャンパなどはないので，ハードウェア的には何も設定する必要はありません．デバイス用のPCのPCIスロットに，評価ボードをそのまま挿し込みます．

次に，ファームウェアを走らせますが，Pcikit.exeは英語モードのDOS上で動作するので，あらかじめus.batを実行してDOSを英語モードに切り替えておきます．その後にPcikit.exeを実行すると，図4の画面が表示されます．これは，Pcikit.exeがPCIバス上に，評価ボードが存在することを確認し，ボード上のISP1582のチップIDを取得して，正常に動作していることを認識した後，初期化処理を行っていることを意味しています．ファームウェアは，この後はISP1582に対して，何もアクセスを行わず，ISP1582からの割り込みイベントを待っている状態になります．

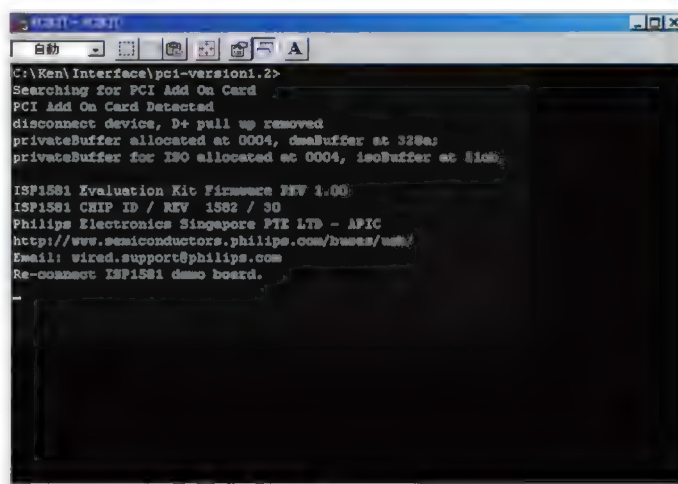


図4　サンプル・ファームウェア起動画面

● エニュメレーションのようす

ここで，ホスト側のPCと，評価ボードの載ったデバイス側のPCをUSBケーブルで接続してみます．すると，ホスト側PCはデバイスが接続されたことを認識し，バス・リセットを発生後，エニュメレーション(enumeration)を始めます．初めて接続する際には，評価キット用のドライバをインストールするための画面が表示されるので，ここでPhkit.exeをおいてあるフォルダを指定し，ドライバをインストールします．

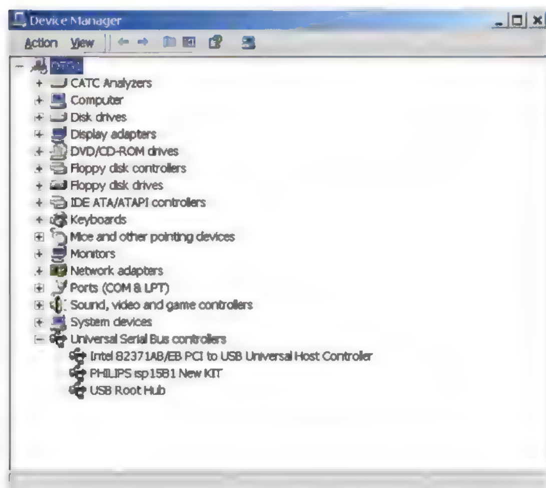


図5　ホスト側のUSBデバイスの認識のようす

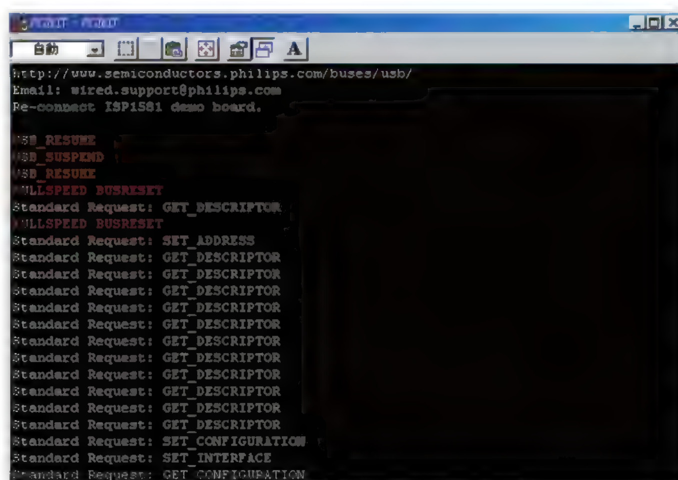


図6　デバイス側の表示画面

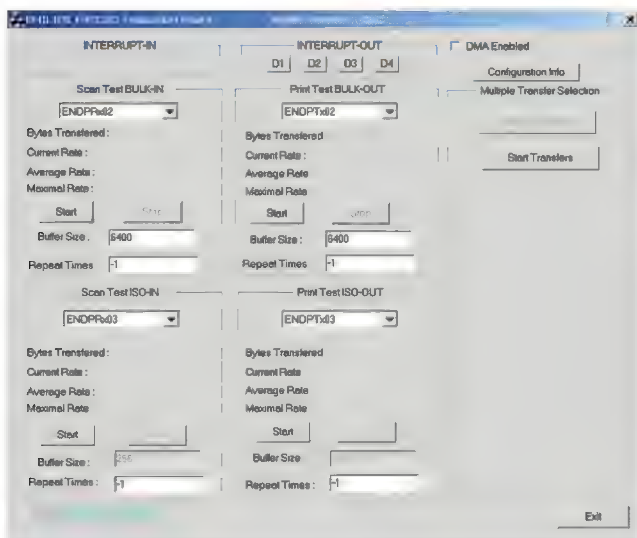


図7　ホスト側でのサンプル・アプリケーション起動のようす

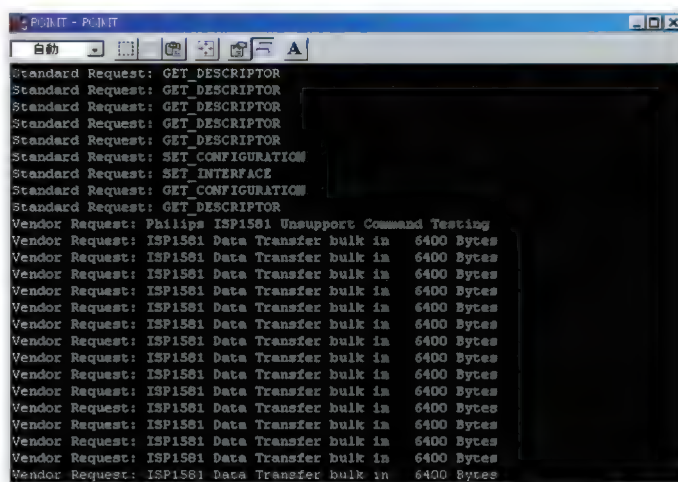


図8　サンプル・アプリケーションによるバルク・イン転送のようす

ドライバのインストール完了後にデバイス・マネージャを確認してみると，図5のように"PHILIPS isp1581 New KIT"と表示されます．実は，インストールしたドライバは一世代前のデバイス・コントローラISP1581用に開発されたものなのですが，同じドライバでISP1582も動作可能なので，そのまま使用しているというわけです．

ここでデバイス側PCを見てみると，図6の画面が表示されています．GET_DESCRIPTOR，SET_ADDRESS，SET_CONFIGURATION，SET_INTERFACE，GET_CONFIGURATIONなどが表示されていて，エニュメレーション処理が行われたことが確認できます．

次に，ホスト側PCで評価用アプリケーションUsbDevice.exeを実行すると図7の画面が表示されます．アプリケーション上に，インタラプト，バルク，アイソクロナス転送のテスト項目が表示されているのがわかります．

この評価キットでは，インタラプト転送に関してはファームウェア側での実装は行っていないので，このテストを行うことはできませんが，そのほかの転送テストは可能です．BULK-INのStartボタンを押すと，デバイス側PCのISP1582からホスト側PCに対して，バルク・インのデータの転送が開始され，転送レートが表示されます．デバイス側PCでも，図8のようにバルク・イン転送が行われているむねが表示されます．ほかの転送テストのStartボタンを押すと，同様にそれぞれの転送が行われることが確認できます．

評価キットには，ファームウェアのソース・コードも付いてくるので，これを基にユーザ自身の手で新規にファームウェアを開発することも可能です．

表2　ISP1582のレジスタ・セット一覧

名　称	ディスティネーション	アドレス	定　義	サイズ（バイト）
初期化レジスタ				
Address	デバイス	00h	USBデバイス・アドレス&イネーブル	1
Mode	デバイス	0Ch	パワー・ダウン・オプション，グローバル割り込みイネーブル，ソフト・コネクト	1
Interrupt Configuration	デバイス	10h	割り込みソース，トリガ・モード，出力プライオリティ	1
OTG	デバイス	12h	OTG	1
Interrupt Enable	デバイス	14h	割り込みソース・イネーブル	4
データ・フロー・レジスタ				
Endpoint Index	エンドポイント	2Ch	エンドポイント選択，データ・フロー・ディレクション	1
Control Function	エンドポイント	28h	エンドポイント・バッファ・マネジメント	1
Data Port	エンドポイント	20h	エンドポイントFIFOデータ・アクセス	2
Buffer Length	エンドポイント	1Ch	パケット・サイズ・カウンタ	2
Buffer Status	エンドポイント	1Eh	エンドポイント・バッファ・ステータス	1
Endpoint MaxPacketSize	エンドポイント	04h	最大パケット・サイズ	2
Endpoint Type	エンドポイント	08h	エンドポイント・タイプ選択	2
DMAレジスタ				
DMA Command	DMAコントローラ	30h	DMA転送制御	1
DMA Transfer Counter	DMAコントローラ	34h	DMA転送バイト・カウント設定	4
DMA Configuration	DMAコントローラ	38h	DMAコンフィグレーション	1
DMA Hardware	DMAコントローラ	3Ch	エンディアン，マスタ/スレーブ，DACK/DREQ/DIOW/DIOR極性選択	1
DMA Interrupt Reason	DMAコントローラ	50h	DMA割り込み	2
DMA Interrupt Enable	DMAコントローラ	54h	DMA割り込みイネーブル	2
DMA Endpoint	DMAコントローラ	58h	エンドポイントFIFO選択，データ・フロー・ディレクション	1
DMA Burst Counter	DMAコントローラ	64h	DMAバースト・カウンタ	2
ジェネラル・レジスタ				
Interrupt	デバイス	18h	割り込みソース	4
Chip ID	デバイス	70h	プロダクトIDコード&ハードウェア・バージョン	3
Frame Number	デバイス	74h	USBフレーム・ナンバ	2
Scratch	デバイス	78h	スクラッチ	2
Unlock Device	デバイス	7Ch	レジスタ・ロック解除	2
Test Mode	PHY	84h	DP&DMステート，内蔵トランシーバ・テスト	1

4 ファームウェアの概要

●ISP1582のレジスタ概要

表2がISP1582のレジスタ・セットです．Initialization Registersで，ISP1582の初期化を行います．これらを使用してISP1582のハードウェア的な動作モードを決定します．

USBデータの送受信は，Data Flow Registersを使って行います．これらは，エンドポイントに対するコマンドとなります．これらのレジスタを使用する際には，まずEndpoint Indexレジスタで該当するエンドポイントを指定してから，ほかのData Flowレジスタにアクセスするという方式をとっています．Data Portレジスタを使用して，Endpoint用FIFOにアクセスを行います．

データの送受信にDMA転送を使う場合には，DMA Registersで制御を行います．DMA信号の極性，動作モードはDMA ConfigurationとDMA Hardwareレジスタで設定します．これは，ISP1582の初期化のときにいっしょに行っておきます．

ISP1582では，USB2.0規格で規定されているテスト・モードの機能をもっています．テスト・モードで動作させるためには，General RegistersのTest Modeレジスタで制御を行います．

●ISP1582の初期化

それでは，ファームウェアのソースを使って，ISP1582の制御手順を確認します．リスト1が，ISP1582の初期化の部分です．これは，`Pcikit.exe`を実行して，PCIバス上の評価ボードを認識し，ISP1582のチップIDを取得した直後に実行されるコードです．

最初にSetAddressEnableを実行することにより，USBアドレスをエニュメレーション時に使用する0に設定します．同時に，アドレス・レジスタのDEVENビットをセットし，ホストからのSetupパケットを受信可能にします．次にSetModeで，ISP1582の動作モードを設定します．ここでは，グローバルの割り込みを許可して，D+のプルアップをONにして，DMAブロックへのクロックの供給を行います．

リスト 1　初期化ルーチン

```
void ISP1581_Initiate(void)
{
    ISP1581_SetAddressEnable(0x00, 0x01);       // set address to 0 and enable it.
    ISP1581_SetMode(reg_mode_set_init_value);
    ISP1581_SetTestMode(0);                     // clear test mode
    ISP1581_SetIntConfig(0x0
                    |intcfg_cdbgmod_as          // control interrupt on ack
                    |intcfg_ddbgmodin_a         // IN data interrupt on ack
                    |intcfg_ddbgmodout_asy      // OUT data on ack and nyet intertupt
                    );
    globe_variable.enabled_intlow = int_busreset|int_susp
                              |int_resume|int_hs_stat
                              |int_dma|int_vbus
                              |int_ep0set
                              |int_ep0rx|int_ep0tx
                              |int_ep1rx|int_ep1tx
                              |int_ep2rx|int_ep2tx;
    globe_variable.enabled_inthigh = int_ep3rx|int_ep3tx
                              |int_ep4rx|int_ep4tx
                              |int_ep5rx|int_ep5tx
                              |int_ep6rx|int_ep6tx
                              |int_ep7rx|int_ep7tx;
    ISP1581_SetIntEnableLow(globe_variable.enabled_intlow);
    ISP1581_SetIntEnableHigh(globe_variable.enabled_inthigh);
    ISP1581_ConfigEndpoint();
    ISP1581_SetDMAHDCfg(dmahd_dreqpolh|dmahd_eotpolh);
    ISP1581_SetDMAConfig(dmacfg_modediorw);
}
```

SetTestModeでは，テスト・モードをクリアして，通常動作を可能にしています．デフォルトでは通常は動作モードになっているので，実際にはこのコードは冗長ですが，テスト・モードではないことを明示的にするために，入れてあります．SetIntConfigでは，エンドポイントが，どのような動作をしたときに割り込みを発生させるかを設定します．

コントロール・エンドポイントとイン・エンドポイントではACKで，アウト・エンドポイントではACKとNYETで割り込みが発生します．その次のSetIntEnableLowとSetIntEnableHighでは，それぞれの割り込み要因を許可しています．ConfigEndpointでは，使用しないエンドポイントを含めて，すべてのエンドポイントのMaxパケット・サイズと方向を指定して，その後に使用するエンドポイントをイネーブルにしています．評価キットでは，DMA転送もサポートしているので，最後にSetDMAHDCfgとSetDMAConfigで，DMA信号の極性と，動作モードを決定しています．この後，ファームウェアは，ISP1582からの割り込みイベントを待つことになります．

リスト 3　エンドポイント 4と 5の処理例

```
void Isr_Ep04rxDone(void)
{
    ISP1581_IntClearh(int_ep4rx);
    ISP1581_ReadBulkEndpoint(EPINDEX4EP04OUT, &ui_gTEST2, 1);
    return;
}

void Isr_Ep05txDone(void)
{
    ISP1581_IntClearh(int_ep5tx);
    ISP1581_WriteBulkEndpoint(EPINDEX4EP05IN, &ui_gTEST1, 1);
    return;
}
```

● 割り込みルーチン

リスト 2が割り込みルーチンになります．まず最初に行うことはUnlockDeviceです．ISP1582のレジスタはサスペンド時には，アクセスできないようにロックされています．レジュームまたはバス・リセットにより，ISP1582がサスペンドから動作状態に入った後もこのロック状態が続き，レジスタの読み書きはできません．

この状態を解除するのが，UnlockDeviceです．レジュームまたはバス・リセットにより，割り込みルーチンがコールされる場合，最初にUnlockDeviceを行っておかないと，ファームウェアが，割り込み要因を知るためにInterruptレジスタを読むことができないので，この処理が必要です．

次に，割り込み要因を特定するためにReadInterruptRegisterLowとReadInterruptRegisterHighを実行し，InterruptEnableレジスタの値とANDを行って許可している，割り込み要因のみを処理するようにします．各割り込み要因ビットは，該当するビットに1を書き戻すことでクリアされます．この操作を行うのが，IntClearl，IntClearhになります．

リスト 3が，エンドポイント 4と 5の処理例を表しています．エンドポイント 4はバルク・アウトとして使用しています．ここでの処理は非常に簡単で，割り込み要因をクリアして，FIFOの内容を指定したメモリ空間に読み出しているだけです．同様にバルク・インのエンドポイント 5では，割り込み要因をクリアして，次に送信するデータをFIFOに書き込んでいます．

エンドポイント 0に対しては，SETUP，IN，OUT用に，別のFIFOが割り当てられており，それぞれに対して，割り込み要因が用意されています．コントロール転送が連続する，エニュメレーション時には，この特徴によりステータス・ステージを

リスト 2　割り込みルーチン

```
void fn_usb_isr(void)
{
    USHORT   int_low, int_high;

    bISP1581flags.bits.At_IRQL1 = 1;
    ISP1581_UnlockDevice();
    int_low = (ISP1581_ReadInterruptRegisterLow() & globe_variable.enabled_intlow); // mask unwanted interrupt.
    int_high = (ISP1581_ReadInterruptRegisterHigh() & globe_variable.enabled_inthigh);

    if(int_low != 0 || int_high != 0)
    {
        if(int_low & int_busreset)
        {
            ISP1581_UnlockDevice();
            ISP1581_IntClearl(int_busreset);
            if(int_low & int_hs_stat)
            {
                bISP1581flags.bits.ConnectSpeed = HighSpeed;
                Isr_BusReset();
                textattr(0x09);
                wprintf( W_MESSAGE, WHITE, "¥r¥nHIGHSPEED BUSRESET");
                textattr(0x07);
                int_low &= ~int_hs_stat;
                ISP1581_IntClearl(int_hs_stat);
            }
            else
            {
                bISP1581flags.bits.ConnectSpeed = FullSpeed;
                Isr_BusReset();
                textattr(0x04);
                wprintf( W_MESSAGE, WHITE, "¥r¥nFULLSPEED BUSRESET");
                textattr(0x07);
            }
        }

        if(int_low & int_hs_stat)
        {
            ISP1581_UnlockDevice();
            ISP1581_IntClearl(int_hs_stat);
            if(bISP1581flags.bits.ConnectSpeed == FullSpeed)
            {
                bISP1581flags.bits.ConnectSpeed = HighSpeed;
                textattr(0x09);
                wprintf( W_MESSAGE, WHITE, "¥r¥nSET HIGHSPEED");
                Isr_BusReset();
            }
            else
            {
                textattr(0x09);
                wprintf( W_MESSAGE, WHITE, "¥r¥nHIGHSPEED RESTORED");
            }
            textattr(0x07);
        }

        if(int_low & int_vbus)
            Isr_VbusON();
        if(int_low & int_susp)
            Isr_SuspendChange();
        else if (int_low & int_resume)
            Isr_Resume();
        if(int_low & int_dma)
            Isr_DmaEot();
        if(int_low & (int_sof|int_psof))
            Isr_SOF();

        if(int_low & int_ep1rx)
            Isr_Ep01rxDone();
        if(int_low & int_ep1tx)
            Isr_Ep01txDone();
        if(int_low & int_ep2rx)
            Isr_Ep02rxDone();
        if(int_low & int_ep2tx)
            Isr_Ep02txDone();
        if(int_high & int_ep3rx)
            Isr_Ep03rxDone();
        if(int_high & int_ep3tx)
            Isr_Ep03txDone();
        if(int_high & int_ep4rx)
            Isr_Ep04rxDone();
        if(int_high & int_ep4tx)
            Isr_Ep04txDone();
        if(int_high & int_ep5rx)
            Isr_Ep05rxDone();
        if(int_high & int_ep5tx)
            Isr_Ep05txDone();
        if(int_high & int_ep6rx)
            Isr_Ep06rxDone();
        if(int_high & int_ep6tx)
            Isr_Ep06txDone();
        if(int_high & int_ep7rx)
            Isr_Ep07rxDone();
        if(int_high & int_ep7tx)
            Isr_Ep07txDone();
        if(int_low & int_ep0tx)
            Isr_Ep00TxDone();
        if(int_low & int_ep0rx)
            Isr_Ep00RxDone();
        if(int_low & int_ep0set)
            Isr_EP0Setup();
    }
    bISP1581flags.bits.At_IRQL1 = 0;
}
```

リスト 4　リクエストの実行(DeviceRequest_Handler)

```
void DeviceRequest_Handler(void)
{
    UCHAR type, req;

    type = ControlData.DeviceRequest.bmRequestType & USB_REQUEST_TYPE_MASK;
    req = ControlData.DeviceRequest.bRequest & USB_REQUEST_MASK;

    if ((type == USB_STANDARD_REQUEST) && (req < MAX_STANDARD_REQUEST))
         (*StandardDeviceRequest[req])();
    else if ((type == USB_CLASS_REQUEST) && (req <= MAX_CLASS_REQUEST))
         (*ClassDeviceRequest[req])();
    else if ((type == USB_VENDOR_REQUEST) && (req   < MAX_VENDOR_REQUEST))
         (*VendorDeviceRequest[req])();
    else{
         Chap9_StallEP0();
    }
}
```

リスト 5　標準リクエストをコールするテーブル

```
#define MAX_STANDARD_REQUEST    0x0D
code void (*StandardDeviceRequest[])(void) =
{
    Chap9_GetStatus,
    Chap9_ClearFeature,
    Chap9_StallEP0,
    Chap9_SetFeature,
    Chap9_StallEP0,
    Chap9_SetAddress,
    Chap9_GetDescriptor,
    Chap9_StallEP0,
    Chap9_GetConfiguration,
    Chap9_SetConfiguration,
    Chap9_GetInterface,
    Chap9_SetInterface,
    Chap9_StallEP0
};
```

リスト 6　クラス・リクエストとベンダ・リクエストのテーブル

```
#define MAX_VENDOR_REQUEST    0x0F
code void (*VendorDeviceRequest[])(void) =
{
    EnableIsoMode,
    ISP1581Bus_ControlEntry,
    reserved,
    reserved,
    reserved,
    reserved,
    reserved,
    reserved,
    reserved,
    reserved,
    reserved,
    reserved,
    read_write_register,
    reserved,
    reserved,
    reserved
};

#define MAX_CLASS_REQUEST    0x00
code void (*ClassDeviceRequest[])(void) =
{
    ML_Reserved
};
```

示す0レングス・パケットが，次のコントロール転送のSETUPパケットに上書きされてしまうことなく，各転送を確実に完了させることが可能です．

● リクエストの実行

SETUP用のFIFOに受信したパケットは，割り込みルーチン内で解析され，その後にメイン・ループ内で，各リクエストに対する処理が行われます．それがリスト 4に示す DeviceRequest_Handlerになります．この中では，SETUPパケットのbmRequestTypeフィールドを確認して，標準リクエスト，クラス・リクエスト，ベンダ・リクエスト用の関数をコールしています．

リスト 5が標準リクエストをコールするためのテーブルになります．それぞれの関数は，USB規格のデバイス・フレームワーク(USB規格の第9章)で規定されたとおりの動作をするように記述されています．独自のファームウェアを設計する際には，この部分には変更を加えずに，リスト 6に示すクラス・リクエストとベンダ・リクエストのテーブルに関数名を記述して，新しくその関数を定義します．

評価キットでは，ベンダ・リクエストをサポートしており，3種類のリクエストが用意されていることがわかります．Pcikit.exeでは，これらのリクエストを実行して，転送バイト数を確認し，対応する転送モードがバルク・イン，バルク・アウト，アイソクロナス・イン，アイソクロナス・アウトのどれかを決定して，ホスト側PCのUsbDevice.exeとアプリケーション・レベルでの通信を行います．USB通信は，基本的に割り込みベースで行われるので，メイン・ループで行われるのは，リクエストの実行のほかには，ISP1582をサスペンド状態にするための処理程度です．

5 USB学習キットとして動作させる

● USB学習キット相当の仕様

実験的にデータ転送を行うのであれば，クラス・リクエストやベンダ・リクエストを実装せずに，それぞれのエンドポイントにアクセスして，インタラプト，バルク，アイソクロナス転送を行うことも可能です．ここでは，ISP1582 PCI評価キットを，参考文献 1)で紹介されているUSB学習キットと同様の動作をするように，ファームウェアを変更してみます．

ホスト側PCのドライバとアプリケーションは，USB学習キット用のものを，そのまま使用することにします．オリジナルの学習キットには，8個のLEDと8個のディップ・スイッチ，1個のプッシュ・スイッチがついていますが，ISP1582 PCI評価キットに，それらを実装するのは難しく，またPCの中のスロットに挿して使うので，実装しても操作や確認ができません．

そこでディップ・スイッチとプッシュ・スイッチの代わりにキーボードからのキー入力を使用し，LED表示の代わりにファームウェア実行時のDOS上の画面にどのLEDが点灯されたかを表示することにします．したがって，PCI評価キットのハードウェアには何も変更は加えません．

● USB学習キットのエンドポイント仕様

アプリケーションは，3本のエンドポイントを使って，ター

表3　USB学習キットのエンドポイントの仕様

エンドポイント0	コントロール・エンドポイント(USB標準)
エンドポイント1	インタラプトIN(プッシュ・スイッチ入力)
エンドポイント2	未使用
エンドポイント3	未使用
エンドポイント4	バルクOUT(LED出力)
エンドポイント5	バルクIN(ディップ・スイッチ入力)

ゲットと通信を行うように設計されています．表3はアプリケーションが，ターゲットに要求するエンドポイントの構成です．エンドポイント2と3は未使用になっていますが，オリジナルのUSB学習キットでは，これら2本をプログラムのダウンロード制御に使っていてエンドポイント・デスクリプタで定義しておかないとアプリケーションが動作しないので，合計5本のエンドポイントをエンドポイント・デスクリプタとして定義してあります．

ファームウェアに変更を加えて実行し，エニュメレーションを行った際のホストPC側のデバイス・マネージャでは，図9のようにUUSBD用USBデバイスの下のTEST USB Deviceとして認識されます．

図10(a)は，学習キットのホスト側PCのアプリケーションを実行したときの画面です．LEDのそれぞれのビットにチェックを入れてLEDボタンを押すと，学習キット上の対応するLEDが点灯します．DIP SWボタンを押すと，学習キット上のディップ・スイッチの状態が表示されます．また，学習キットのプッシュ・スイッチを押すと図10(b)の画面が表示されることになります．

デバイス側PCのファームウェアの画面には，LED，ディップ・スイッチ，プッシュ・スイッチの状態を表示するための項目を追加しました．図11がその画面です．

● ファームウェアの変更部分

リスト7が使用するエンドポイントのデスクリプタ情報になります．エンドポイント1のインタラプト転送は，10msごとに行われるように設定されています．エンドポイント4と5のバルク転送は，アプリケーション上のLEDとDIP SWボタンが押されたときにのみ行われます．このアプリケーションは簡単な転送テストを目的として作成されているので，コントロール転送を使ったベンダ・リクエストは使用せずに，ISP1582の対応するエンドポイントに単純にアクセスして，通信を行います．

今回，ファームウェアの変更は必要最小限にとどめています．グローバル変数，ui_gTEST1，ui_gTEST2，ui_gTEST3を新しく定義し，それぞれをディップ・スイッチ，LED，プッシュ・スイッチとして割り当ててホスト側PCとの通信に使用しています．

画面表示は，メイン・ループで新しくコールされるui_status_monitorで，それぞれの変数に変化があった場合に表示が更新されるようになっています．キー入力は同様にメイ

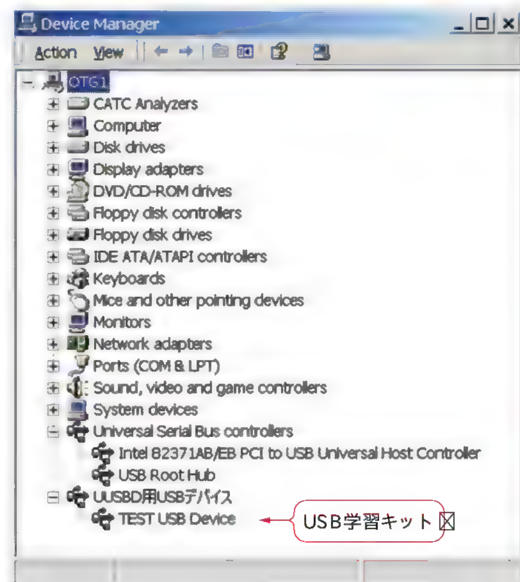
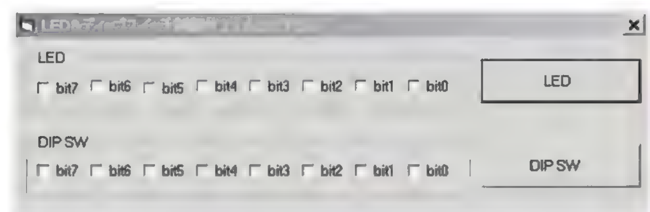
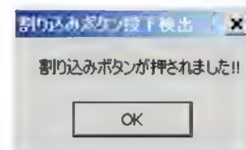


図9　USB学習キットを接続したときのデバイス・マネージャの表示



(a) LED＆ディップ・スイッチ設定画面



(b) 割り込みボタン押下検出ダイアログ

図10　USB学習キットのホスト側アプリケーションのようす

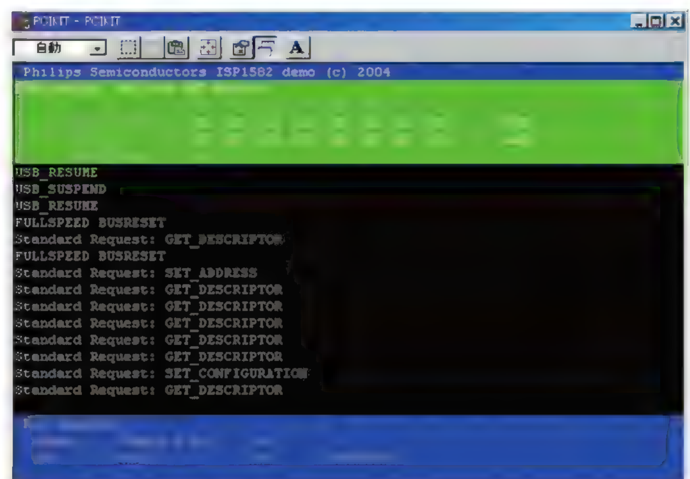


図11　デバイス側の画面

リスト 7　エンドポイントのデスクリプタ情報

```
USB_ENDPOINT_DESCRIPTOR EP_Descr_ALTER[NUM_ENDPOINTS_OF_ALTER] =
{
    {
        sizeof(USB_ENDPOINT_DESCRIPTOR),
        USB_ENDPOINT_DESCRIPTOR_TYPE,
        0x81,
        USB_ENDPOINT_TYPE_INTERRUPT,
        SWAP(maxepsize_16),
        10
    },
    {
        sizeof(USB_ENDPOINT_DESCRIPTOR),
        USB_ENDPOINT_DESCRIPTOR_TYPE,
        0x02,
        USB_ENDPOINT_TYPE_BULK,
        SWAP(maxepsize_FS),
        0
    },
    {
        sizeof(USB_ENDPOINT_DESCRIPTOR),
        USB_ENDPOINT_DESCRIPTOR_TYPE,
        0x83,
        USB_ENDPOINT_TYPE_ISOCHRONOUS,
        SWAP(maxepsize_256),
        1
    },
    {
        sizeof(USB_ENDPOINT_DESCRIPTOR),
        USB_ENDPOINT_DESCRIPTOR_TYPE,
        0x04,
        USB_ENDPOINT_TYPE_BULK,
        SWAP(maxepsize_FS),
        0
    },
    {
        sizeof(USB_ENDPOINT_DESCRIPTOR),
        USB_ENDPOINT_DESCRIPTOR_TYPE,
        0x85,
        USB_ENDPOINT_TYPE_BULK,
        SWAP(maxepsize_FS),
        0
    }
};
```

リスト 8　メイン・ループのキー入力部

```
void virtural_switch( void )
{
    if ( kbhit() )                          //  get key hitting
    {
        unsigned char k;

        k    = getch();                     //  retrieve content of key buffer

        switch ( k )                        //  do the action for user
        {
            case '8' :
                {
                    ui_gTEST3 = True;       // Push button pressed
                }
                break;

            case '7' :
            case '6' :
            case '5' :
            case '4' :
            case '3' :
            case '2' :
            case '1' :
            case '0' :

                //  action for the bit toggling
                //  the key hits from [0] to [7] overwrites "ui_gTEST1" variable.
                //  "ui_gTEST1" is for DIP-SW.

                {
                    unsigned char n;
                    n    = ( k - '0');

                    wprintf( W_MESSAGE, WHITE, "DIP switch toggle command : bit=%c   %s¥r¥n", k, ((ui_gTEST1 >> n) & 1) ? "1
                                                                                           --> 0" : "0 --> 1" );

                    ui_gTEST1 = ((ui_gTEST1 >> n) & 1) ? (ui_gTEST1 & ~(1 << n)) : (ui_gTEST1 | (1 << n));
                }
                break;

            default :
                break;
        }
    }
}
```


New Products――ルネサステクノロジ，大容量のフラッシュ・メモリを内蔵した160MHzの車載用32ビットRISCマイコンを発売
(株)ルネサステクノロジは，大容量のフラッシュ・メモリを内蔵した，動作周波数が160MHzの車載用32ビット・マイコン「M32192グループ」を12月よりサンプル出荷する．自動車の安全走行支援システム機器などでの用途を想定している．

ン・ループでコールされるvirtual_switch(リスト8)で検出され，ui_gTEST1(ディップ・スイッチ)，ui_gTEST3(プッシュ・スイッチ)が更新されます．これらの値は，割り込みルーチン内で，対応するエンドポイントに書き込まれ，ホスト側PCのアプリケーションに送信されます．

また，LED制御情報は，割り込みルーチン内でエンドポイント4に受信した値を，ui_gTEST2に読み出すことにより取得します．各割り込みルーチンをリスト9に示します．

デバイス側PCのキーボードから，0～7までの数字を入力すると，画面のディップ・スイッチの対応するビットがトグルし，ホスト側PCアプリケーションのDIP SWボタンを押すと，バルク・インで，その情報がホスト側PCに転送され，表示されることが確認できます．同様に，バルク・アウトを使って，LED制御情報がデバイス側PCに転送され，表示されることも確認できます．

また，プッシュ・スイッチはデバイス側PCからの'P'キーの入力があると，インタラプト・イン転送により，ホスト側PCで「割り込みボタンが押されました!!」というダイアログが表示されることが確認できます〔図10(b)〕．

以上のようにファームウェアの簡単な変更により，ISP1582 PCIキットが，USB学習キットと同様の動作をするようになることがわかります．これは，USB規格の第8章をISP1582が，第9章をファームウェアが完全に実装しているためで，その上位層で動作するファームウェアは，比較的簡単に実装できることを意味しています．

リスト9　各割り込みルーチン

```
void Isr_Ep01txDone(void)    // INT button status
{
    ISP1581_IntClearl(int_ep1tx);
    ISP1581_WriteBulkEndpoint(EPINDEX4EP01IN, &ui_gTEST3, 2);
    ui_gTEST3 = False;
    return;
}

void Isr_Ep04rxDone(void)    //   LED control
{
    ISP1581_IntClearh(int_ep4rx);
    ISP1581_ReadBulkEndpoint(EPINDEX4EP04OUT, &ui_gTEST2, 1);
    return;
}

void Isr_Ep05txDone(void)    //   DIP switch status
{
    ISP1581_IntClearh(int_ep5tx);
    ISP1581_WriteBulkEndpoint(EPINDEX4EP05IN, &ui_gTEST1, 1);
    return;
}
```

まとめ

今回とりあげた，ISP1582はハイ・スピードUSBデバイス・コントローラですが，今年9月に発売が開始されるハイ・スピードUSB　On-The-Goコントローラ(ISP1761)の中にもISP1582と完全互換のコアが採用されています．ISP1582用のファームウェアを設計しておけば，後はホスト・スタックとOn-The-Go(OTG)制御のファームウェアを設計するだけで，OTG対応が可能になるという利点があります．

ISP1582の情報とサンプル・ファームウェアの入手は，下記URLから参照可能です(サンプル・ファームウェアはユーザ登録後にダウンロード可能)．

▶ISP1582の情報は
http://www.semiconductors.philips.com/cgi-bin/pldb/pip/isp1582.html

▶ファームウェアに関して
http://www.semiconductors.philips.com/buses/usb/products/download/index.html

参考文献

(1) USBハード&ソフト開発のすべて，TECH I Vol.8，CQ出版(株)

ひがしやま・けん　日本フィリップス(株)

SuperHシリーズおよびH8SシリーズのUSBターゲット機能

2 SuperH & H8Sマイコンを使ったUSB機器の開発事例

音堂 栄良/池谷 貴之

ルネサステクノロジのSuperHシリーズやH8Sシリーズに内蔵されているUSBターゲット機能には，エンドポイントの仕様や最大パケット・サイズが異なるだけで，基本的には同じレジスタ構成のターゲット・コントローラが内蔵されている．よって必要なシステム規模に合わせて，SuperHシリーズやH8Sシリーズを選択できる．ここではSuperHシリーズのSH7727をメインに，USBターゲット機能について解説する．（編集部）

はじめに

ここでは(株)ルネサステクノロジの32ビットRISCマイコンSuperH，および16ビット・マイコンH8Sに搭載しているUSBターゲット・モジュール(以後ではファンクション・モジュールと呼ぶ)について解説します．

SuperHおよびH8Sマイコンに搭載されたUSBファンクション・モジュールは，USB2.0規格で規定された標準コマンドのほとんどをハードウェアで自動処理するという特徴をもっています．そのため，USBに詳しくなくても，簡単にUSBシステムを実現できます．

1 USBファンクション・モジュール仕様について

● エンドポイント構成とFIFO仕様

SuperHとH8S搭載USBファンクション・モジュール仕様を解説するにあたり，SH7727とSH7720を例にとって説明します．SH7727，SH7720搭載のUSBファンクション・モジュールは，USB2.0フル・スピードに対応しています．

表1に示すように，SH7727はコントロール転送1チャネル，バルク転送2チャネル，インタラプト転送1チャネルを内蔵しています．また，コントロール転送以外のすべてのエンドポイントはConfiguration#1－Interface#0－AlternateSetting#0に配置されています．このエンドポイント構成をもったデバイスはSH7727，SH7705，SH7641，H8S/2212，H8S/2218，H8S/2212，

表1　エンドポイント構成

エンドポイント名	名称	エンドポイント番号（from Host）	転送タイプ	最大パケット・サイズ	FIFO容量
エンドポイント 0	EP0s	エンドポイント #0	セットアップ	8バイト	8バイト
	EP0i	エンドポイント #0	コントロール・イン	8バイト	8バイト
	EP0o	エンドポイント #0	コントロール・アウト	8バイト	8バイト
エンドポイント 1	EP1	エンドポイント #1	バルク・アウト	64バイト	128バイト
エンドポイント 2	EP2	エンドポイント #2	バルク・イン	64バイト	128バイト
エンドポイント 3	EP3	エンドポイント #3	インタラプト・イン	8バイト	8バイト

(a) SH7727のエンドポイント構成

エンドポイント名	名称	エンドポイント番号（from Host）	転送タイプ	最大パケット・サイズ	FIFO容量
エンドポイント 0	EP0s	エンドポイント #0	セットアップ	8バイト	8バイト
	EP0i	エンドポイント #0	コントロール・イン	8バイト	8バイト
	EP0o	エンドポイント #0	コントロール・アウト	8バイト	8バイト
エンドポイント 1	EP1	エンドポイント #*N*	バルク・アウト	64バイト	128バイト
エンドポイント 2	EP2	エンドポイント #*N*	バルク・イン	64バイト	128バイト
エンドポイント 3	EP3	エンドポイント #*N*	インタラプト・イン	8バイト	8バイト
エンドポイント 4	EP4	エンドポイント #*N*	アイソクロナス・アウト	64バイト	128バイト
エンドポイント 5	EP5	エンドポイント #*N*	アイソクロナス・イン	64バイト	128バイト

(b) SH7720のエンドポイント構成

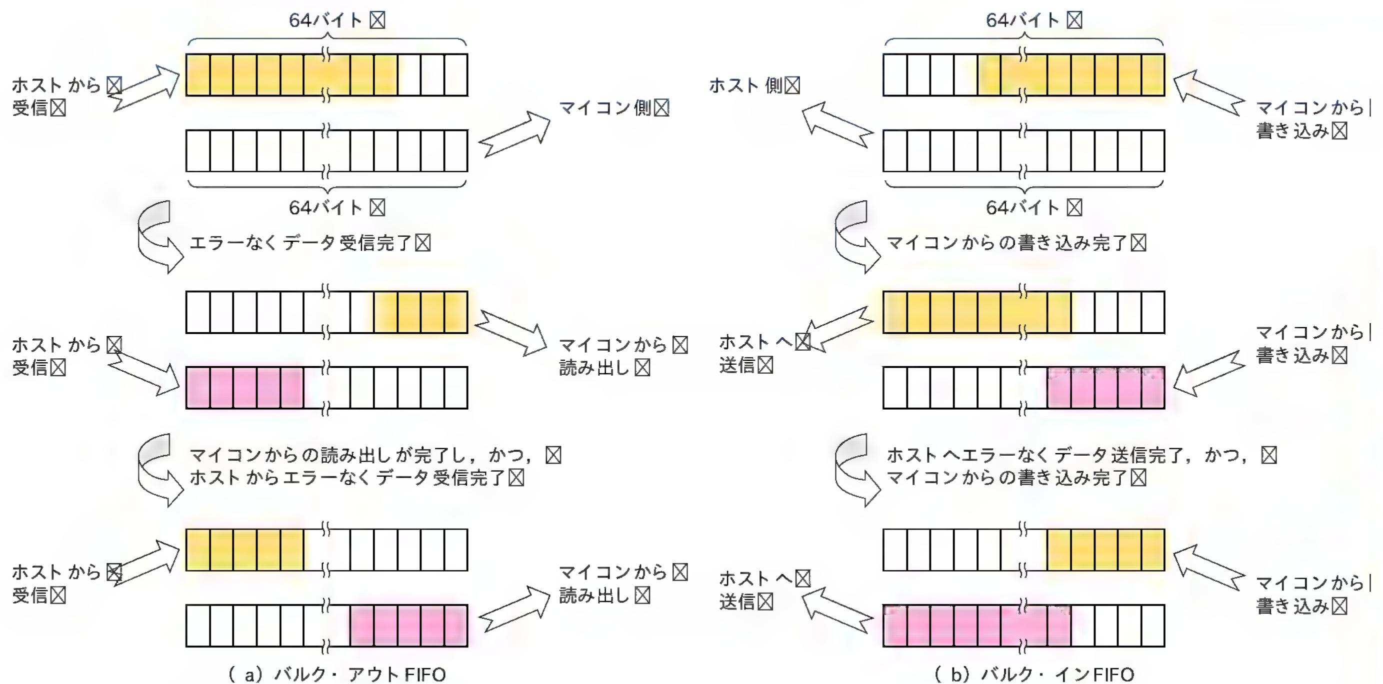


図1 SH7727のエンドポイント構成

H8S/2218はバルク・エンドポイント番号，最大パケット・サイズが異なる）となります．

SH7720，SH7630，SH7660，H8S/2215は，アイソクロナス転送もサポートし，Interface-AlternateSetting番号，ホストから見た際のエンドポイント番号をエンドポイント情報レジスタによってソフトウェアで変更できる機能をもっています．

SH7630，SH7660，H8S/2215は，SH7720と比較し最大パケット・サイズやサポート・エンドポイント数が増えています．詳細は，各マイコンのハードウェア・マニュアルを参照してください．

すべてのマイコンに共通して，コントロール転送，インタラプト転送FIFOは1面構成になっていますが，バルク転送FIFOはデータ伝送効率を考慮して2面構成になっています．バルク・アウト転送の場合，ホストからエラーなくデータ・パケットを受信すると，2面FIFOが切り替わり，割り込みが発生します．マイコンはこの割り込みをもとに受信したデータを読み出すことができます．マイコンでデータを読み出している間，もう一面のFIFOは並行してUSBホストからデータを受信することができます．マイコンによってすべてのデータの読み出しが完了し，かつ，ホストからのデータの受信が完了すると，2面FIFOが切り替わり再び割り込みが発生し，2回目に受信したデータの読み出しが可能となります．バルク・インFIFOも同様な構造となっており，効率よくデータ転送を行うことができます．

一見複雑そうに見受けられますが，マイコンからは1面しか見えていないため，ソフトウェアは2面あることを意識する必要はありません（図1）．

● エラー判定などの処理は自動的にハードウェアで処理

ハードウェアは，データ受信を行う際，受信したデータにエラー（CRC，ビット・スタッフィング，バイト・バウンダリ，最大パケット・サイズなど）があるか否かを判定しています．また，受信するエンドポイントのFIFOに空きがあるか否かも判定しています．

まず，OUTトークンを受信し，FIFOに空きがなければNAKハンドシェークを自動で返信し，その際のデータ・パケットは破棄します．FIFOに空きがあれば受信を開始し，受信したデータにエラーがあった場合，ハンドシェークの返信を行わず，受信したデータを破棄します．

エラーがなかった場合，受信したデータを有効データと判断し，ACKハンドシェークを返信し，そのデータをマイコンで読み出すことができます．また，USB規格で規定されたデータPIDビットも監視しており，データPIDが誤っていると，ACKハンドシェークを返信し，受信したデータを破棄します．データを送信する場合も同様に，ホストからのハンドシェークを監視し，ホストからACKハンドシェークを受信すると送信したFIFOを空にし，ACKハンドシェークを受信しないとFIFO内にデータを保持します．

以上すべてのエラー判定，FIFO管理をハードウェアで行うため，ソフトウェアはデータ・エラーをまったく意識することなくデータの送受信が可能です．

● USB標準コマンド処理とUSBステート管理

USB規格では表2のような標準コマンドが規定されています．SuperHおよびH8S搭載のUSBファンクション・モジュール

は，Get_Descriptor，Set_Descriptor，Synch_Frame以外のすべての標準コマンドをハードウェアで自動処理します．そのため，ユーザ・ソフトウェア設計者は，USB規格の細かい仕様をあまり意識しなくてもUSBシステムを実現させせることができます．これは，Configuration，Interface，Alternate Settingと各エンドポイントの管理，Haltの管理，USBステートの管理をハードウェアが行っているためです．クラス・コマンド，ベンダ・コマンドはすべてソフトウェアにて処理します．

USB規格では，デバイス・ステートが定義されています．デバイス・ステートとしては，Powered，Default，Address，Configuredが定義されており，バス・リセット，Set_Addressコマンド，Set_Configurationコマンドによってステート遷移する必要がありますが，このデバイス・ステートもすべてハードウェアが自動管理しています．そのため，ソフトウェアはデバイス・ステートを意識する必要がありません．

表2　USB標準コマンド処理

標準コマンド名	データ・ステージ	コマンド概要	処　理
Clear_Feature	なし	デバイスの特定機能をクリア	ハードウェア
Get_Configuration	イン	現在のコンフィグレーション値を報告	ハードウェア
Get_Descriptor	イン	ディスクリプタ情報を報告	ソフトウェア
Get_Interface	イン	指定したインターフェースのAlternateSetting値報告	ハードウェア
Get_Status	イン	デバイスの状況を報告	ハードウェア
Set_Address	なし	アドレスの設定	ハードウェア
Set_Configuration	なし	コンフィグレーション値を設定	ハードウェア
Set_Descriptor	アウト	ディスクリプタ情報の変更	ソフトウェア
Set_Feature	なし	デバイスの特定機能を設定	ハードウェア
Set_Interface	なし	指定したインターフェースのAlternateSetting値設定	ハードウェア
Synch_Frame	アウト	エンドポイントの同期フレームを設定	ソフトウェア

● USB割り込みとソフトウェア処理

表3にSH7727のUSB割り込みを示します．そのほかSuperHマイコン，H8Sマイコンに関しても，割り込み名称やビット名は異なる場合がありますが，基本的にはまったく同意の割り込み機能をもっています．

2 初期化処理とコントロール転送について

ここで各割り込みごとに必要なソフトウェア処理と，注意事項に関して説明します．各デバイスのハードウェア・マニュアルに記載されているフローチャートも同時に参照してください．

● 初期化処理

まず，ソフトウェア初期化処理においてUSBクロックやピン・ファンクション・コントローラの設定が完了したら，PULLUP_Eビットや汎用I/Oポートを用いてUSB信号D+ラ

表3　SH7727のUSB割り込み

割り込み名	初期値	R/W	意　味
BRST(バス・リセット)割り込み	0	R/W	USBバス上でバス・リセット信号を検出したとき，1にセットされる
EP1 FULL(FIFOフル)割り込み	0	R	エンドポイント1がホストから1パケット分のデータを正常に受信するとセットされ，FIFOバッファに有効データが存在する間1を保持する．このビットはステータス・ビットのため，クリアはできない
EP2 TR(転送リクエスト)割り込み	0	R/W	ホストからエンドポイント2に対するイン・トークンを受信したとき，FIFOバッファに有効な送信データが存在しない場合にセットされる．FIFOバッファにデータを書き込んでパケット送信イネーブルをセットするまで，ホストに対してNAKハンドシェークを返す
EP2 EMPTY(FIFOエンプティ)割り込み	1	R	エンドポイント2の2面構成の送信用FIFOバッファのうちの少なくとも1面が送信データを書き込める状態であるときにセットされる．ステータス・ビットのため，クリアはできない
SETUP TS(セットアップ・コマンド受信完了)割り込み	0	R/W	エンドポイント0がアプリケーション側でデコードする必要のあるセットアップ・コマンドを正常に受信し，ホストにACKハンドシェークを返したとき1にセットされる
EP0o TS(受信完了)割り込み	0	R/W	エンドポイント0がホストからのデータを正常に受信してFIFOバッファに格納し，ホストにACKハンドシェークを返したとき1にセットされる
EP0i TR(転送リクエスト)割り込み	0	R/W	ホストからエンドポイント0に対するイントークンを受信したとき，FIFOバッファに有効な送信データが存在しない場合にセットされる．FIFOバッファにデータを書き込んでパケット送信イネーブルをセットするまで，ホストに対してNAKハンドシェークを返す
EP0i TS(送信完了)割り込み	0	R/W	エンドポイント0からホストにデータを送信し，ACKハンドシェークが返ってきたときセットされる
EP3 TR(転送リクエスト)割り込み	0	R/W	ホストからエンドポイント3に対するINトークンを受信したとき，FIFOバッファに有効な送信データが存在しない場合にセットされる．FIFOバッファにデータを書き込んでパケット送信イネーブルをセットするまで，ホストに対してNAKハンドシェークを返す
EP3 TS(送信完了)割り込み	0	R/W	エンドポイント3からホストにデータを送信し，ACKハンドシェークが返ってきたときセットされる
VBUSF(USBバス接続)割り込み	0	R/W	ファンクションがUSBバスに接続されたとき，および切断されたときに1にセットされる．接続/切断の検出には，本モジュールのV_{BUS}端子を使用する

インの1.5kΩをプルアップしてください(H8Sの場合はUSBケーブル接続後プルアップ).

SH7720, SH7630, SH7660, H8S/2215の場合は1.5kΩをプルアップする前にエンドポイント情報レジスタを設定する必要があります. USBケーブルが接続されると, USBケーブル内のV_{BUS}ライン(5V)を用いてLSIのV_{BUS}端子が"L"レベルから"H"レベルに変化し, USBバス接続割り込みが発生します. 本割り込みは, V_{BUS}端子の立ち上がりエッジ変化, および立ち下がりエッジ変化を検出しているため, 接続, および切断ともに検出することができます. ただし, USBケーブルの接続/切断は必ずしもソフトウェアで検出する必要はないため, 各割り込みの許可/非許可を設定できる割り込みイネーブル・レジスタを用いて, 本割り込みを無視することも可能です(H8Sの場合は, 本割り込みでD+の1.5kΩプルアップを許可/非許可を制御する必要がある).

その後, USBホストはデバイスの接続を認識し, USBバス・リセットを発行してきます. USBファンクション・ハードウェアはバス・リセットを検出すると, BRST(バス・リセット)割り込みを発生させるので, ソフトウェアでは本割り込みを検出し, FIFOのクリアや, ソフトウェアでステートを持っている場合の初期化などを行ってください.

以上が初期化時に発生する割り込みと必要な処理です.

● コントロール・イン転送

次に, USBホストからコントロール転送が開始されます. コントロール転送と割り込みの関係を説明する前に, まず, USB規格で規定されたコントロール転送のステージに関して簡単に説明します.

コントロール転送はセットアップ・ステージ, データ・ステージ, ステータス・ステージの3種から構成され, セットアップ・ステージは必ずセットアップ・トークン・パケット, 8バイトのデータ・パケット, ハンドシェークで構成されます. データ・ステージは, デバイスからホストへデータを送信するイン・トランザクションの場合と, ホストからデバイスへデータを送信するアウト・トランザクションの場合, そしてデータ・ステージが存在しない場合の3種類があります.

データ・ステージがイン・トランザクションであれば, ステータス・ステージはアウト・トランザクションとなり, データ・ステージがアウト・トランザクション, またはデータ・ステージが存在しない場合, ステータス・ステージはイン・トランザクションとなります. また, ステータス・ステージのデータ・パケットは必ずNULL(0バイト長)パケットで構成されます.

最初のバス・リセット受信後, 通常, `Get_Descriptor`コマンドが発行されます. ホストから`Get_Descriptor`を示すセットアップ8バイト・データをエラーなく受信するとSETUP TS(セットアップ受信完了)割り込みが発生します.

ソフトウェアは本割り込みを用いて, まず, 本割り込みフラグのクリアとエンドポイント0インとアウトのFIFOをクリア(FIFOクリア・レジスタを使用)した後, EP0sデータ・レジスタから受信した8バイト・データを読み出し, どのコマンドを受信したかデコードします. USBのメモリ・バッファはFIFO構造になっているため, 8バイトを読み出す場合はEP0sデータ・レジスタから8回読み出しを行う必要があります.

読み出しが完了したら, ハードウェアに対し, 次のデータ・ステージまたはステータス・ステージへ移行してもよいことを知らせるため, トリガ・レジスタのEP0s RDFN(読み出し完了)ビットへ1を書き込みます. また, コントロール・イン転送の場合, 本割り込み処理中でEP0iTR(転送リクエスト)割り込みを禁止にします.

`Get_Descriptor`コマンドではソフトウェアは次にデータ・ステージがイン・トランザクションであることは理解しているはずです. そのため, エンドポイント0のインFIFOを用いてDescriptor情報をホストへ返信させます. 最初, エンドポイント0インFIFOは必ず空になっているはずなので, SETUP TS(セットアップ受信完了)割り込み内でEP0iデータ・レジスタへ送信すべきデータを書き込みます.

ここのでの注意事項としては, EP0iデータ・レジスタへの書き込み単位は, 必ず最大パケット・サイズ以下にする必要があります. ただし, たとえば, 最大パケット・サイズが8バイトで, 送信すべきデータが23バイトであった場合, 1回目8バイト, 2回目8バイト, 3回目7バイトのように, 最後のデータ以外はすべて最大パケット・サイズで送信する必要もあります.

FIFOへの書き込みが完了したら, FIFO内のデータをホストへ送信してもよいことをハードウェアへ知らせるため, トリガ・レジスタのEP0iPKTE(パケット・イネーブル)ビットへ1を書き込んでください. ホストからイン・トークンが発行されると, 先ほど書き込んだFIFO内のデータが自動的にホストへ送信されます. ホストはこのデータをエラーなく受信するとACKハンドシェークを送信します. USBファンクション・モジュールはACKハンドシェークを受信すると, 送信が正常に完了し, エンドポイント0インFIFOが空になったことを知らせるためEP0iTS(送信完了)割り込みを発生させます. ソフトウェアは, この割り込みを用いて残りのデータを先ほどと同じ手順で準備し, ホストへデータを送信させます. これらの動作を繰り返し, 送信すべきデータをすべてホストへ転送させます.

ここでの注意事項としては, ソフトウェアでデータの準備を行っている間にも, USBホストからはイン・トークンが頻繁に送られてきます. そのため, FIFO内にまだ送信すべきデータが存在しない状態でイン・トークンを受信すると, EP0iTR(転送リクエスト)割り込みが発生してしまいます. データ・ステージがイン方向である場合, 転送リクエスト割り込みは使用しないように注意してください. 具体的には, 転送リクエスト割り込みビットが「1」になっていてもソフトウェアでは無視するとともに, 割り込みイネーブル・レジスタは非許可の状態にしておく必要があります.

次に，USBホストは，データ・ステージにおいてすべてのデータを受信し終えるとNULL(0バイト長)パケットを送信してステータス・ステージへ移行させます．このパケットをエラーなく受信するとEP0oTS割り込みが発生し，ソフトウェアはこの割り込みによって一連のコントロール転送が正常に完了したことを知ることができます．具体的には，本割り込みフラグをクリアし，トリガ・レジスタのEP0o RDFN(読み出し完了)ビットへ1を書き込んで終了してください．

以上で，`Get_Descriptor`処理は完了となります．コントロール・イン転送の注意事項としては，USBホストはいつでもデータ・ステージを中断してステータス・ステージへ移行することができる点があります．すなわち，セットアップの8バイト・データで指定したデータ・サイズすべてをホストが受信していなくても，急遽データ・ステージを完了し，ステータス・ステージへ移行される可能性があります．ソフトウェアはこのような場合においても問題なく動作するようにコーディングしなければなりません．

最初の`Get_Descriptor`コマンドが完了すると，2回目のバス・リセットが発行され，その後，`Set_Address`，`Get_Descriptor`，`Set_Configuration`コマンドが発行されます．`Set_Address`，`Set_Configuration`コマンドはハードウェアが自動処理するため，SETUP TS(セットアップ受信完了)割り込みが発生することなく，ソフトウェアは特別な処理を行う必要がありません．ただし，バス・パワードをサポートしたデバイスでは，V_{BUS}からの消費電流をコントロールできるように`Set_Configuration`コマンドを正常に受信したことを知らせる割り込み機能があります．`Set_Configuration`コマンドが完了後，コントロール以外のエンドポイントが使用可能となります．

● コントロール・アウト転送

次にデータ・ステージがアウト方向となるコントロール・アウト転送に関して説明します．コントロール・アウト転送の場合も，まずはセットアップ・ステージから開始されます．セットアップ8バイト・データをエラーなく受信すると，SETUP TS(セットアップ受信完了)割り込みが発生します．ソフトウェアは本割り込みを用いて，まず，本割り込みフラグのクリアとエンド・ポイント0インとアウトのFIFOをクリア(FIFOクリア・レジスタ使用)した後，EP0sデータ・レジスタから受信した8バイト・データを読み出します．その後，何のコマンドを受信したかデコードし，トリガ・レジスタのEP0s RDFN(読み出し完了)ビットへ1を書き込みます．また，コントロール・アウト転送の場合，本割り込み処理中でEP0i TR(転送リクエスト)割り込みフラグをソフトウェアでクリアした後，EP0i TR(転送リクエスト)割り込みを許可してください．

その後，ホストからはエンドポイント0に対しアウト・パケットが発行されます．ハードウェアはトリガ・レジスタのEP0s RDFN(読み出し完了)ビットへ1が書き込まれるまでNAKハンドシェークを返信し続けます．1が書き込まれると，エンドポイント0アウトFIFOは空であるためデータを受信します．エラーなく正常にデータを受信すると，ホストへACKハンドシェークを返信し，EP0o TS(正常受信)割り込みが発生します．ソフトウェアは本割り込みを検出し，本割り込みフラグをクリアした後，EP0o受信データ・サイズ・レジスタを読み出して，何バイトのデータを受信したかを認識します．受信したデータ・サイズ分のEP0oデータ・レジスタから受信したデータをすべて読み出してください．

読み出しが完了したら，最後に次のパケットを受信してもよいことをハードウェアに知らせるため，トリガ・レジスタのEP0o RDFN(読み出し完了)ビットへ1を書き込んでください．この動作により，次の新しいパケット受信し再度EP0o TS(正常受信)割り込みが発生します．ちなみに，トリガ・レジスタのEP0o RDFN(読み出し完了)ビットへ1を書き込むまでは，ハードウェアが自動でNAKハンドシェークを返信しています．

ソフトウェアはこれらの動作を繰り返します．ホストから送信されるすべてのデータを受信してください．ホストはデータ・ステージにおいてすべてのデータを送信し終えると，エンドポイント0に対しイン・トークンを発行してきます．エンドポイント0インFIFO内にまだ送信すべきデータが存在しない状態なので，イン・トークンを受信するとEP0i TR(転送リクエスト)割り込みが発生します．ソフトウェアはこの割り込みをもとに，本割り込みフラグをクリアし，データ・ステージからステータス・ステージに移行されたことを認識し，NULL(0バイト長)パケットを送信させます．

また，コントロール・アウト転送の場合，本割り込み処理中でEP0i TR(転送リクエスト)割り込みを禁止にしてください．エンドポイント0インFIFOへデータを書き込むことなく，トリガ・レジスタのEP0i PKTE(パケット・イネーブル)ビットへ1を書き込むと，NULL(0バイト長)パケットを送信する準備が完了し，ホストからイン・トークンが発行されるとNULL(0バイト長)パケットが自動的にホストへ送信されます．ホストはこのデータをエラーなく受信すると，ACKハンドシェークを送信します．

USBファンクション・モジュールはACKハンドシェークを受信すると，送信が正常に完了したことを知らせるためEP0i TS(送信完了)割り込みを発生させます．ソフトウェアは，この割り込みを用いてすべてのコントロール・アウト転送が正常に完了したことを認識することができます．

3 バルク転送について

● バルク・アウト転送

コントロール転送はUSB規格でステージが規定されているため，少し複雑に感じられたかもしれませんが，バルク転送は非常にシンプルな処理となります．

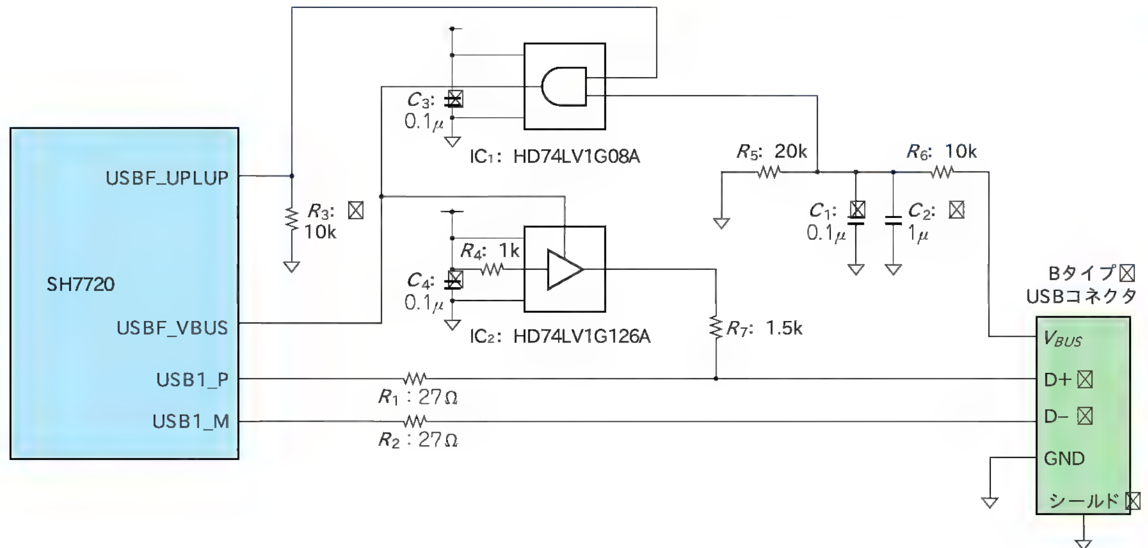


図 2
USB 外部回路図例
(SH7720)

まずは，バルク・アウト転送に関して説明します．ホストからバルク・アウト・パケットをエラーなく受信するとホストへACK ハンドシェークを返信し，EP1 FULL(FIFOフル)割り込みが発生します．ソフトウェアは本割り込みによりエンドポイント 1バルク・アウト FIFOにデータを受信したことを認識します．

本割り込み処理で，まずは何バイトのデータを受信したかを認識するため，コントロール転送のときと同様に，EP1受信データ・サイズ・レジスタを読み出し，受信したデータ・サイズ分 EP1データ・レジスタから受信したデータをすべて読み出してください．

読み出しが完了したら，トリガ・レジスタの EP1 RDFN(読み出し完了)ビットへ 1を書き込んで割り込みを完了してください．基本的にソフトウェアは，この処理を繰り返すだけでデータを受信することができます．

バルク・アウトの FIFOは 2面構造になっていますが，ソフトウェアからは 1面しかないように見えるので 2面構造を意識する必要がありません．また，EP1 FULL(FIFOフル)割り込みはステータス・フラグであり，ソフトウェアから 0を書き込んでクリアすることはできません．データをすべて読み出し，EP1 RDFN(読み出し完了)ビットへ 1を書き込んだ際，もう一面の FIFOに受信したデータが存在しないとハードウェアで自動的にクリアされます．もう一面の FIFOにすでにデータを受信完了していると再度割り込みが発生します．

● バルク・イン転送

次にバルク・イン転送に関して説明します．ソフトウェアにおいてホストへ送信したいデータが発生した場合，EP2 EMPTY (FIFOエンプティ)割り込みを許可してください．すると，最初は FIFOが空であるため即座に割り込みが発生します．ソフトウェアはこの割り込みで EP2データ・レジスタを介してエンドポイント 2バルク・イン FIFOへデータを書き込みます．

注意事項としては，1回の書き込みは最大パケット・サイズである 64バイト以下にする必要があります．データをすべて書き込み終えると FIFO内のデータをホストへ送信しても良いことをハードウェアへ知らせるため，トリガ・レジスタの EP2 PKTE (パケット・イネーブル)ビットへ 1を書き込んでください．

ホストからイン・トークンが発行されると先ほど書き込んだ FIFO内のデータが自動的にホストへ送信されます．

まだ送信したいデータが存在している場合，このまま割り込み処理を完了してください．エンドポイント 2バルク・イン FIFOにデータを書き込んでよい状態になると，再度 EP2 EMPTY(FIFOエンプティ)割り込みが発生するので，送信したいデータをすべて送信し終えるまで EP2データ・レジスタへの書き込みとトリガ・レジスタの EP2 PKTE(パケット・イネーブル)ビットへ 1書き込みを繰り返してすべてのデータを送信してください．

もう送信すべきデータが存在しなくなった場合，最後の割り込み処理で EP2 EMPTY(FIFOエンプティ)割り込みを禁止にしてください．ほとんどのアプリケーションでは EP2 EMPTY (FIFOエンプティ)割り込みのみでデータの送信が可能です．必要であれば EP2 TR(転送リクエスト)割り込みを使用してホストからのイン・トークン要求を確認することが可能ですが，バルク・イン FIFOへデータを書き込む際は，必ず EP2 EMPTY(FIFOエンプティ)割り込みを使用してください．NULL(0バイト長)パケットを送信したい場合は，FIFOにデータを書き込まないでトリガ・レジスタの EP2 PKTE(パケット・イネーブル)ビットへ 1を書き込んでください．

4 USB 外部回路

ここでは，SH7720を例にして USB の外部回路を説明します．ただし，本回路は参考回路であり，すべてのシステムにおいて

動作保証するものではありません．また，外部回路は使用するLSIによって多少異なります．必ずハードウェア・マニュアルの外部回路例を参照してください．

● D＋/D－ラインの処理

図2(p.77)にSH7720のUSBファンクションの外部回路を示します．USB信号線D+およびD−には必ず直列抵抗(R_1，R_2)が必要となります．SH7720の場合は27Ωですが，デバイスによって値が異なるので，各デバイスのハードウェア・マニュアルを確認してください．抵抗値を誤ると波形が乱れて正常に通信できない場合があります．

直列抵抗(R_1，R_2)はできるだけLSIの近くに配置し，抵抗からUSBコネクタまでは差動インピーダンス90Ωで同長に配線することが望ましいです．外部からのサージおよびESD対策が必要なシステムでは，低容量の高速通信用サージ保護ダイオードなどで保護してください．基本的にコイルの使用はすすめませんが，どうしても必要な場合，波形をできるだけ乱さないコイルを選択してください．D+をプルアップする1.5kΩ(R_7)はD+ラインの近くに配置してください．

● V_{BUS}ラインの処理

V_{BUS}ラインにはコンデンサ(C_1，C_2)を付加しています．これは，USB規格で1～10μFを付加するよう記載されているためです．また，コンデンサがあるとノイズの影響を受けにくくなるという効果もあります．USBケーブル非接続時にV_{BUS}ラインがフローティングしないようにプルダウン抵抗(R_5)が必要です．V_{BUS}ラインは電源であり，入力IC(HD74LV1G08A)を保護する観点から，分圧抵抗(R_5，R_6)でV_{BUS}電圧を下げていますが，分圧した際の電圧がIC(HD74LV1G08A)のV_{ih}電圧を十分超える値に設定する必要があります．

USBケーブル接続からD+を1.5kΩでプルアップさせるまでの遅延時間は最大100msにしなければならないとUSB規格で規定されているため，RC遅延時間が100ms以内になるようR_5，R_6，C_1，C_2を調整してください．R_5，R_6の値を決める際，USBサスペンド時のV_{BUS}消費電流最大500μAを守る必要があります．

V_{BUS}が"L"レベルのときD+の1.5kΩはプルアップしてはならないとUSB規格で規定されているため，AND回路IC$_1$(HD74LV1G08A)を使用してください(H8Sの場合，ソフトウェアでAND)．システム電源がOFFのとき，USBケーブルがPCに接続されV_{BUS}が"H"レベルになる可能性があるので，IC$_1$はLSI電源がOFFであっても入力電圧印加可能なトレラントICを使用してください．

また，本ラインはRC回路で構成されているため信号がゆっくり変化します．そのためIC$_1$はヒステリシスをもったICを選択することをおすすめします．SH7720のV_{BUS}端子が通信中に"L"レベルになるとハードウェアはUSB規格で規定されたパワード・ステートへ移行してしまうので，V_{BUS}ラインにノイズが載らないよう配線/基板レイアウトには注意してください．

● そのほか

D+の1.5kΩプルアップを制御する汎用I/Oポートは，パワーON初期値としてハイ・インピーダンスとなるポートを使用してください．

コネクタのシールドは筐体のGNDまたはシステムのGNDに接続してください．ノイズの影響がでる場合は，電源用コイルなどで対策してください．

5 サンプル・プログラム

次にサンプル・プログラムの概要を解説します．ここで解説するサンプル・プログラムはSH7720用のプログラムで，Solution Engine Light MS7720RP01(日立超LSIシステムズ製，以下Solution Engineと表記)にポーティングされています．なお，本サンプル・プログラムは，すべてのシステムにおいて動作を保証するものではありません．

● USB Communicationクラス

サンプル・プログラムはUSB Communication Class <Abstract Control Model Serial Emulation>(以下，COMクラス)に対応しています．PC側のデバイス・ドライバとしてUSBモデムなどで使用されるWindows 2000，Windows XP標準搭載COMポート・エミュレーション・ドライバ(usbser.sys，以下COMクラス・デバイス・ドライバ)を使用します．

COMクラスに対応したUSBデバイスをホストとなるPCに接続すると，COMクラス・デバイス・ドライバが呼ばれ，COMポートAPIを持った仮想COMポートとして認識されます(COMポートが追加される)．ホストPCのユーザ・アプリケーションからはシリアル通信に使用するCOMポートAPIを介してデバイスにアクセスすることができます(図3)．

サンプル・プログラムではSolution Engine上にあるCOMポートを使用してUSBとシリアルの変換を行っています(図4)．

● データ転送概要

サンプル・プログラムではUSB-シリアル

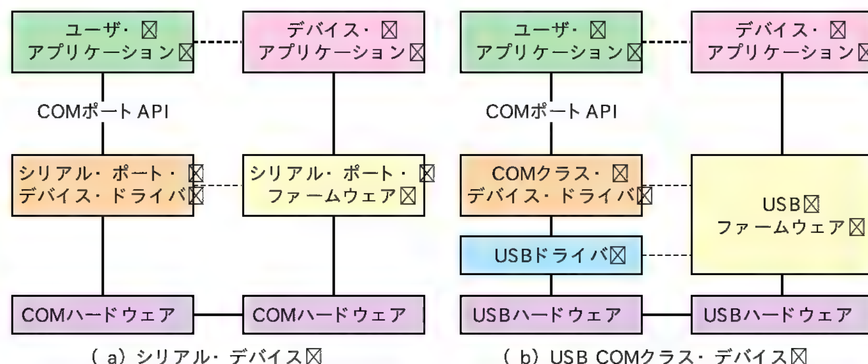


図3 シリアル・デバイスとCOMクラス・デバイス

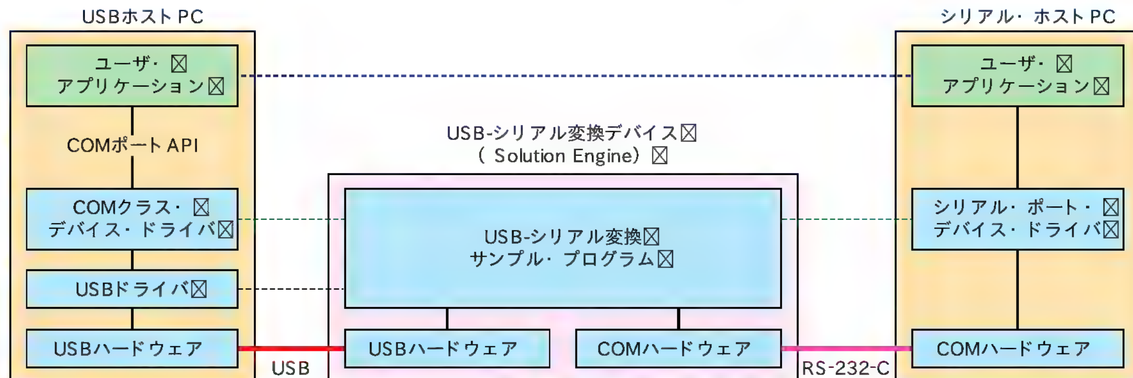


図4　COMクラス・サンプル・プログラム

変換のデータ送受信用としてそれぞれリング・バッファを使用しています．USB バルク・アウト転送でデータを受信すると，割り込みを発生させて USB →シリアル転送で使用するリング・バッファにバルク・アウトで受信したデータを書き込みます．また，シリアルでデータを受信すると割り込みを発生させてシリアル→ USB 転送で使用するリング・バッファにシリアルで受信したデータを書き込みます（図 5）．

割り込みルーチンから復帰した通常ルーチンでは，つねにリング・バッファに送信すべきデータが存在するかを監視しています．USB →シリアル転送で使用するリング・バッファにデータが存在した場合はシリアルで，シリアル→ USB 転送で使用するリング・バッファにデータが存在した場合はバルク・インでデータを出力します．

● COM クラス・コマンド

COM クラスでは，クラス・コマンド（表 4）を使用してシリアル・モジュールの転送速度の変更などを行います．

SET_LINE_CODING コマンドは，調歩同期式データ伝送で使用されるパラメータ（ビット・レート，ビット長，パリティなど）を設定します．サンプル・プログラムでは，SET_LINE_CODING で受信したデータをもとにシリアル・モジュールを再設定します．

GET_LINE_CODING コマンドは，デバイスのシリアル・モジュール設定値を USB ホストが取得する際に使用されます．

SET_CONTROL_LINE_STATE コマンドは制御信号を設定します．サンプル・プログラムでは，このリクエストについてはデコードのみを行います．

クラス・コマンド詳細に関しては Universal Serial Bus Class Definitions for Communication Devices を参照してください．

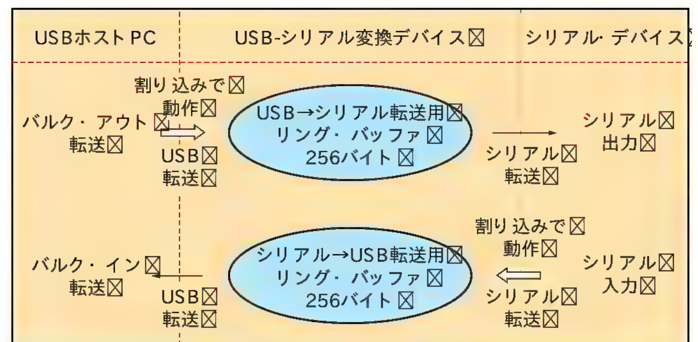


図5　リング・バッファと割り込みの関係

● サンプル・プログラムのファイルと関数

サンプル・プログラムに含まれるファイルを表 5 に，各関数の相関関係を図 6 に示します．

● 使用方法

本サンプル・プログラムの使用方法は次のとおりです．

（1）ファイル SetUsbInfo.h 内の変数 DeviceItem[] に VendorID，ProductID を設定する

（2）ComClass.inf 内の [Models] にも VendorID，ProductID（SetUsbInfo.h と同じ値）を設定し，[Strings] に会社名などの情報を設定する

（3）サンプル・プログラムをコンパイルする．コンパイル時の領域の設定は付属の lnkSet3.sub ファイルを参照

（4）Solution Engine の CN6（シリアル・ポート）とシリアルのホスト PC をクロス・ケーブルで接続する

（5）Solution Engine 上にプログラムをロードする．ルネサスソリューションズ製 E10USB エミュレータを使用する場合は，

表4　COMクラス・コマンド

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_LINE_CODING（20h）	Zero	Interface	8	Line Coding Structure
10100001B	GET_LINE_CODING（21h）	Zero	Interface	8	Line Coding Structure
00100001B	SET_CONTROL_LINE_STATE（22h）	Control Signal Bitmap	Interface	Zero	None

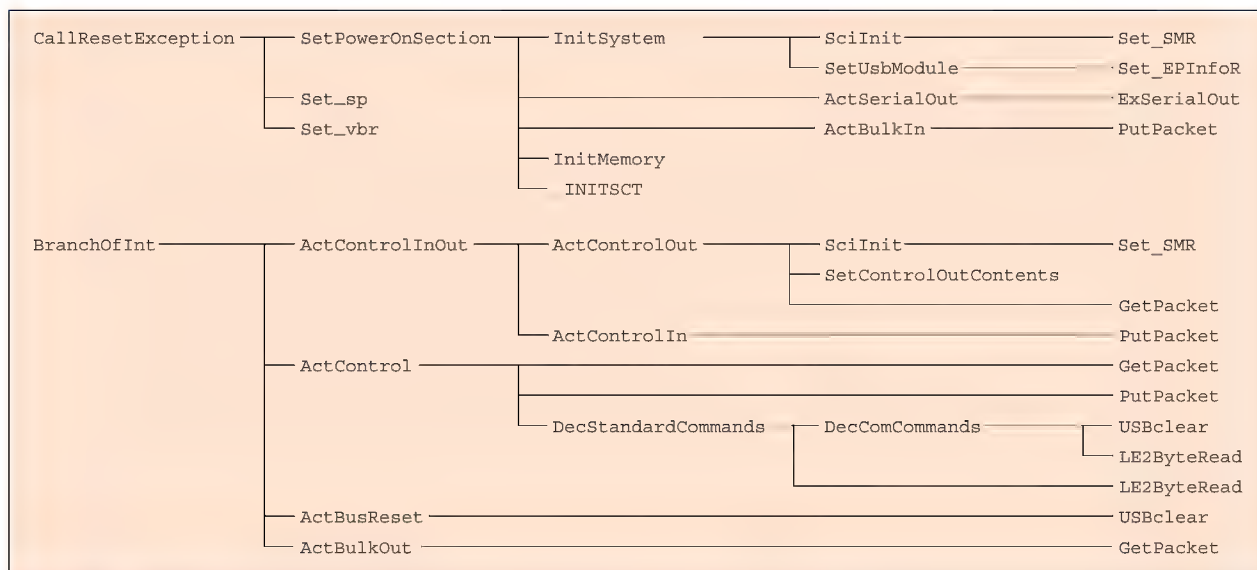


図6　関数の相関関係

表5　サンプル・プログラムの各ファイル

ファイル名	おもな役割
StartUp.c	ベクタ・テーブルの設定，マイコンの初期設定，リング・バッファのクリア
DoSerial.c	シリアル送受信を実行，SCIF0モジュールの制御
UsbMain.c	割り込み要因の判定，パケットの送受信
DoRequest.c	ホストが発行するセットアップ・コマンドの処理
DoControl.c	コントロール転送を実行
DoBulk.c	バルク転送を実行
DoRequestComCommand.c	COMクラス・コマンドの処理
SysMemMap.h	Solution Engineのメモリ・マップのアドレス定義
SetUsbInfo.h	USBの構成を定義
SetMacro.h	マクロ定義
SetSystemSwitch.h	システムの動作設定（現在の設定はビッグ・エンディアン）
SH7720.h	SH7720レジスタ定義
CatTypedef.h	構造体定義
CatProType.h	プロトタイプ宣言
AsmFunction.src	スタック・ポインタの設定
sct.src	変数の初期値設定に使用
lnkSet3.sub	コンパイル時の領域の設定
ComClass.inf	ComClassドライバ・インストール時のinfファイル
7720E10A.hdc	プログラムをダウンロードする際のCPU初期化設定ファイル〔（株）ルネサスソリューションズ製E10USBエミュレータ用〕

事前に7720E10A.hdcを実行させ，Solution Engineの初期設定を行う

（6）PCカウンタを関数CallResetExceptionの先頭にあわせてサンプル・プログラムを実行させる

（7）Solution EngineのCN10（USBタイプBコネクタ）とUSBホストPCをUSBケーブルで接続する

（8）COMクラスのドライバのインストールが始まるのでComClass.infを指定し，ドライバをインストールする

（9）デバイス・マネージャで追加されたCOMポートの番号を確認する．

（10）シリアル・ホストPCでハイパーターミナルを起動し，Solution Engineと接続したポートを選択する（表6を参照して通信速度などのシリアル設定を行う）

（11）USBホストPCでハイパーターミナルを起動し，先ほどデバイス・マネージャで確認したCOMポート番号を選択する．シリアルの設定値はシリアル・ホストPCで設定した値と必ず同じ値に設定すること

（12）ハイパーターミナルどうしで通信が行えることを確認

表6　通信設定

パラメータ	設定可能値
ビット・レート［bps］	9600/19200/38400/57600/115200
データ・ビット	8/7
パリティ	なし/奇数/偶数
ストップ・ビット	1，2
フロー制御	Xon/Xoffのみ

まとめ

本章ではサンプル・プログラムを用いてUSBシリアルを変換するCOMクラスについて解説しました．ルネサステクノロジではSuperH，H8S用としてCOMクラス以外にも，ストレージ・クラス，プリンタ・クラス，HIDクラス（H8Sのみ）などのサンプル・プログラムも準備しています．

おんどう・えいりょう/いけや・あつゆき　（株）ルネサステクノロジ

小規模マイコンにも接続可能なUSBホスト/ターゲット・コントローラ

3 SL811を使った簡易ホストとUSBキーボードの接続実験

桑野 雅彦

第2部ではホスト機能をもったUSBコントローラを取り上げる．組み込み向けでは，ホストとターゲットの両方の機能を内蔵したコントローラもある．ここで取り上げるSL811もその一つで，8ビット・マイコンにも接続可能なホスト機能をもったコントローラである．ここではSL811のホスト機能について詳しく解説したあと，USBキーボードを接続する例について紹介する．（編集部）

はじめに

サイプレスセミコンダクタ社(以下サイプレス)のSL811は，USBのホスト/ターゲットのどちらにもなることができるという，少し変わったUSBコントローラです．前身となるSL11のバージョン・アップ版にあたるデバイスで，ピン配置/機能面とも上位互換になっています．

SL811は，もともとはScanLogic社で作られたものなのですが，サイプレスがScanLogic社を買収したことから，現在はサイプレスの製品となっています．最近登場したEZ-OTGのようなCPUコアは内蔵しておらず，外部のCPUによってコントロールするUSBホスト/ターゲット・コントローラです．

今でこそUSBにもOn-The-Go(OTG)仕様が規定されて，ホスト/ターゲットのどちらにもなれるようなデバイスも出てきていますが，SL811はそれよりも前にできたデバイスなので，デバイス単体ではOTG対応にはなりません．また，SL811はホスト機能があるといっても，その機能，性能はパソコン用のOHCIやUHCIといった本格的なホスト・コントローラとは異なり，ごく基本的なパケットの発行や少量のデータのやり取りが行えるようになっているにすぎません．細々とした伝送制御のめんどうをCPUがみなくてはならないため，多くの機器のめんどうをみようとするとCPUの負荷も高くなりやすく，パフォーマンスもUHCIやOHCIに比べると見劣りすることは否めません．

このように，SL811はホスト・コントローラとはいってもごく小規模なものであり，用途にはかなりの制約があることはまちがいありません．しかし，そのような制約を理解したうえで利用すれば，安価であること，ピン数も少なく，インターフェースも単純でさまざまなCPUに対応可能であるうえ，USBの特徴である電源供給が可能であることや，RS-232Cなどよりもはるかに高速なデータ転送が行えるといったUSBの利点が生きてきます．

たとえば，相手を決め打ちして一対一でしか使わないのであれば，ややこしいプロトコル・スタックなどを考える必要はありません．複数のデバイスのサポートのためのハブ・デバイスの管理(USBではハブも一つのデバイスであるため，複数のデバイスを動作させるためにはハブ・デバイスの管理が必須となる)やアクセスのスケジューリングを行ったり，プラグ＆プレイ実現のためにさまざまなデバイス用のドライバをダイナミックにロード/アンロードするといったややこしい操作も必要ありません．

USBの基本的なプロトコルに基づいて接続，初期化の処理を行った後は，ターゲットのエンドポイントのリード/ライトを行うだけでデータの受け渡しが行えます．RS-232Cでは何かとめんどうなフロー制御やCRCによるエラー検出もUSBならばUSBコントローラに実装済みで，コマンドはエンドポイント0を，データ入力はエンドポイント1を使うといったぐあいに分離して利用することができるというのも利点です．

今回はこのコンパクトな仕様のUSBコントローラであるSL811のホスト機能を使って実際にUSBキーボードとの伝送実験を行ってみることにします．

1 SL811の仕様

SL811の仕様を表1に示します．フル・スピード(12Mbps)，ロー・スピード(1.5Mbps)のデバイスを接続可能なホスト機能を内蔵しています．電源電圧は3.3Vですが，I/Oは5Vトレラントになっているので，5V系のCPUなどとも直結可能です．

パッケージは28ピンのPLCCパッケージ品(SL811HSH)と48ピンのTQFPパッケージ品(SL811HST)が用意されていますが，出ている信号自体は両方とも同じです．

28ピン・パッケージ品があることからもわかるとおり，SL811の外部インターフェースは信号線も少なく，ごくシンプルです．図1に示すようにデータ・バス，チップ・セレクトにリード/ライト，割り込み，DMA関係と，ごくありふれたCPUイン

表1 SL811の仕様

項 目	仕 様
USB準拠規格	USB1.1
USB機能	ホスト/ターゲットのいずれかに切り替え使用(端子で設定)
USBチャネル数	1
伝送速度	12Mbps(フル・スピード)/1.5Mbps(ロー・スピード)
I/Oバス幅	8ビット(5Vトレラント)
バッファRAM	240バイト
サスペンド/レジューム/ウェイクアップ/ロー・パワー・モード	サポート
SOF，CRC5，CRC16生成	自動
電源電圧	3.3V単一
パッケージ	28ピンPLCC(SL811HSH)/48ピンTQFP(SL811HST-AC)

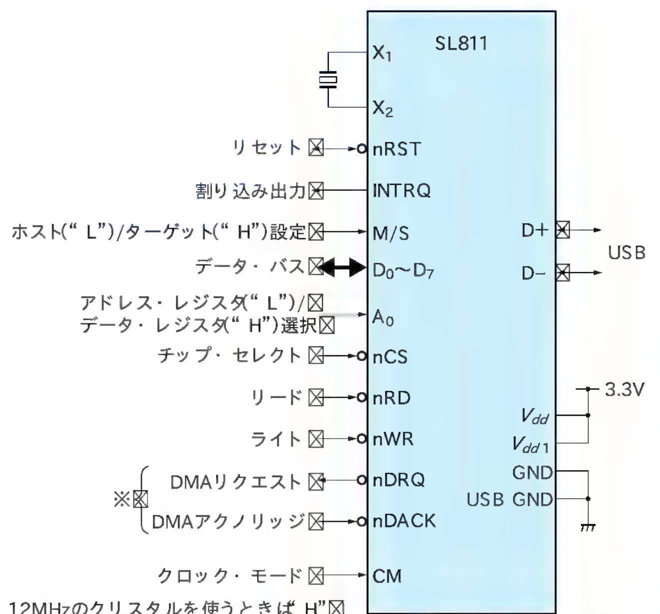


図1 SL811の入出力信号

ターフェースとUSBポートがあるだけです．A_0が"L"レベル(0)のときがアドレス・レジスタ，"H"レベル(1)のときがデータ・レジスタにアクセスするようになっているので，まずA_0を"L"レベルにしてレジスタ番号を書き込み，続いて"H"レベルにしてデータのリード/ライトを行うという手順になります．

2 内部ブロック

SL811の内部ブロックは図2のようになっています．内部の256バイト分の空間にRAMバッファ/コントロール・レジスタが配置されていて，先頭16バイト分がSL811の制御やステータス用などのレジスタです．それ以降の240バイト分が1パケット分のデータ入出力のためのバッファになっています．USB1.1のアイソクロナス伝送の場合，1パケットの最大サイズは1023バイトですが，SL811では239バイト(255−16バイト)までの伝送しか行えない点に注意が必要です．

バッファ領域のアクセスもレジスタと同じ手順で行えますが，まとまったサイズのデータのリード/ライトのために，アドレスのオート・インクリメント・モードがあり，先頭アドレスを指定すればあとはデータ・レジスタへのアクセスだけで自動的にアドレスが更新され，連続領域のリード/ライトが効率よく行えるようになっています．

3 レジスタ・マップ

RAMバッファとコントロール・レジスタのアドレス配置は表2のようになっています．全体で256バイト分の領域があり，このうち`00h`～`0Fh`の16バイト分が内部レジスタ領域，`10h`～`FFh`の240バイトがUSB経由でのデータやデバイス・リクエストなどのパケット・データを納めるバッファ領域になっています．

USB-AとUSB-Bという名称で2組のレジスタがありますが，これは2チャネル分のインターフェースがあるということではありません．デバイス・リクエストの後のデータ・インなど，連続したアクセスを行う場合に最初の動作が終わる前に次の要求をセットできるようにして，転送効率を上げることができるようにしたものです．ちなみに，SL811の前身であるSL11のときにはUSB-B側のレジスタはありませんでした．

この機能を使わない場合にはUSB-A側のレジスタだけの操作でも問題はありません．今回のサンプルでも簡単にするため，

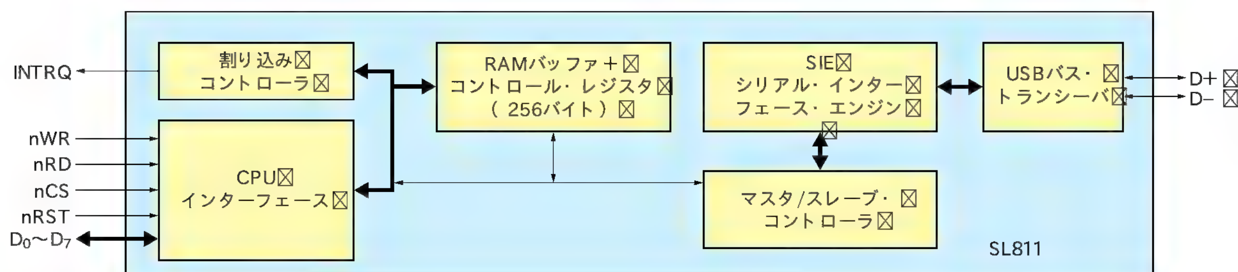


図2 SL811の内部ブロック図

この機能は使わず，USB-A 側のレジスタだけ使っています．

それではこれらのレジスタの詳細を見ていきましょう．

● USBホスト・コントロール・レジスタ(00h/08h)

ビットの配置は図3のようになっています．

▶ Arm

SL811に対して転送開始を指示するビットです．転送するパケット・データ(OUT方向のとき)や，パケットID，ターゲットのUSBアドレスやエンドポイント・アドレスなどを設定した状態でArmとEnableビットをともに'1'にすると指定した転送動作が実行されます．SL811が転送動作を完了すると，Armビットは自動的に'0'にクリアされます．

▶ Enable

SL811の動作イネーブル・ビットです．

スレーブ(USBターゲット)動作のときにこのビットを'1'にするとエンドポイントへのアクセスがイネーブルされます．'0'にするとホストから送られてきたパケットはすべて無視します．Enableが'1'でArmが'0'ならば，ホストからの要求にはNAKで応答します．

ホスト動作の場合，転送を開始するにはArmビットとEnableビットの両方をセットする必要があります．

▶ Direction

SL811からUSBバスへの転送のときにはこのビットを'1'に，USBバス側からSL811への転送のときには'0'にします．今回はホスト動作なので，OUT方向(ホストからターゲット)のときに'1'，IN方向のときに'0'に設定します．

表2 SL811のレジスタ・マップ

レジスタ名称	レジスタ・アドレス
USB-A ホスト・コントロール・レジスタ	00h
USB-A ホスト・ベース・アドレス	01h
USB-A ホスト・ベース・レングス	02h
USB-A ホストPID，デバイス・エンドポイント(Write時)	03h
USB-A ステータス(Read時)	
USB-A ホスト・デバイス・アドレス(Write時)	04h
USBコントロール・レジスタ1	05h
割り込みイネーブル・レジスタ	06h
(予約)	07h
USB-B ホスト・コントロール・レジスタ	08h
USB-B ホスト・ベース・アドレス	09h
USB-B ホスト・ベース・レングス	0Ah
USB-B ホストPID，デバイス・エンドポイント(Write時)	0Bh
USBステータス(Read時)	
USB-B ホスト・デバイス・アドレス(Write時)	0Ch
USB-B コントロール・レジスタ1	0Dh
SOFカウンタ下位(Write時)	0Eh
ハードウェア・レビジョン・レジスタ(Read時)	
SOFカウンタ上位/コントロール・レジスタ2	0Fh
メモリ・バッファ領域	10h～FFh

▶ ISO

アイソクロナス転送のときに'1'にします．それ以外のとき(コントロール転送，バルク転送，インタラプト転送)のときには'0'にします．

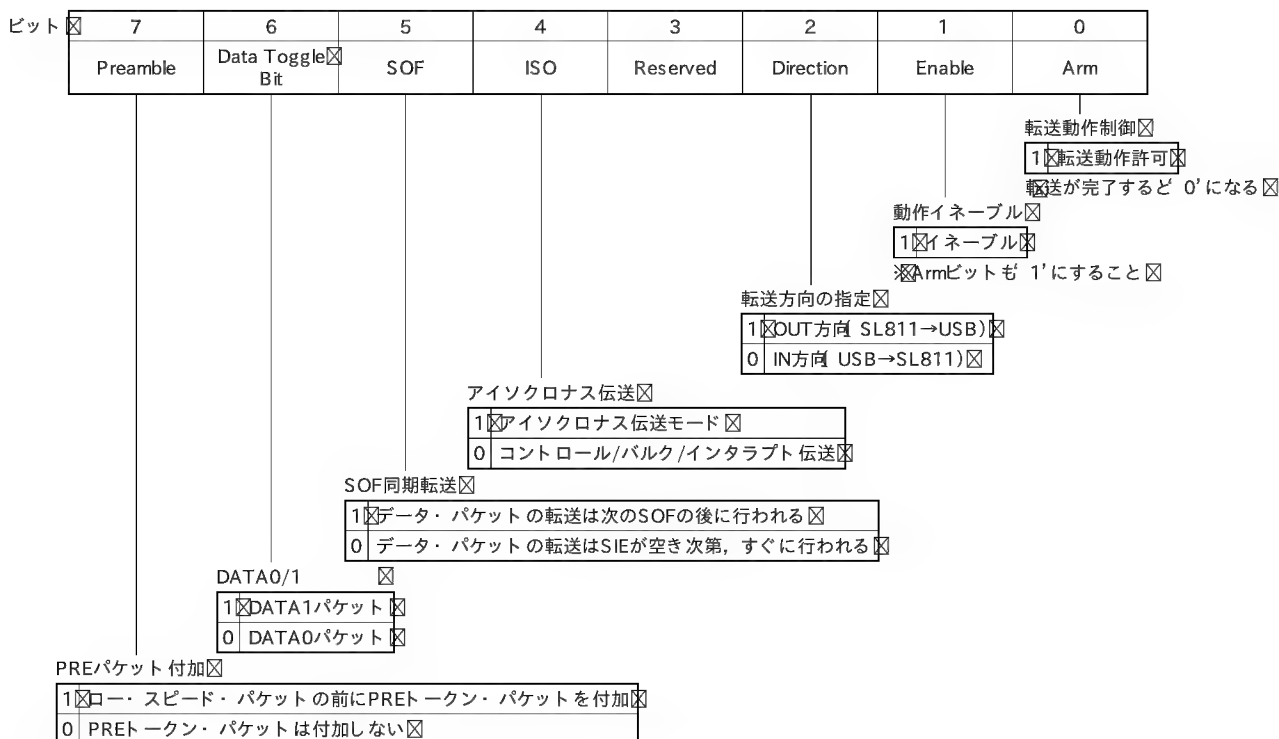


図3 ホスト・コントロール・レジスタの構成

▶ SOF

SOF（Start Of Frame）パケットに同期してデータ転送動作を行うか否かを決めます．USB規格上ではSOFパケットは1ms周期で送出されることになっています．このビットを'1'にすると，要求した転送動作は次のSOFパケットの直後に行われます．'0'の場合には要求した転送動作はSIE（シリアル・インターフェース・エンジン）がアイドル状態であれば即座に行われます．

▶ Data Toggle Bit

DATA0のときには'0'，DATA1のときには'1'にします．IN方向のときにここで指定したDATA0/1と，ターゲットから送られてきたDATA0/1が一致しなければなりません．

▶ Preamble

ハブの先にロー・スピード・デバイスが接続された場合，ロー・スピード・パケットの前にPREパケットを付加することで，ハブに対してロー・スピード・ポートを開かせます．Preambleビットが'1'になっていると，PREパケットが自動的に付加され，続くデータ転送は自動的にロー・スピードで行われます．このときハブとの間のモードはフル・スピード設定なので，レジスタ05hのビット5（USB SPEED）を'0'，レジスタ0Fhのビット6（Data Polarity Swap）を'0'にします．

SL811に直接ロー・スピード・デバイスが接続されているときにはUSB SPEEDビット，Data Polarity Swapビットとも'1'にして直接ロー・スピードで動作させます．このときPreambleビットの設定は無視されます．

● ホスト・ベース・アドレス（01h/09h）/ホスト・ベース・レングス（02h/0Ah）

ホスト・ベース・アドレスは，転送データを格納するSL811内部のバッファRAMの先頭アドレスを指定します．指定されたアドレスはSL811内部RAM領域の先頭になります．

ホスト・ベース・レングスは転送データのデータ長を指定します．

● USBステータス（03h/0Bh）

リード時のみ有効です．ビット配置は図4のとおりです．USB動作関係のステータスがまとめられています．

▶ ACK

転送動作が正常終了すると'1'になります．

▶ Error

転送動作中にエラーが検出されると'1'になります．

▶ Time-Out

スレーブ（ターゲット）が応答せず，タイム・アウトが発生すると'1'になります．

▶ Sequence

USB1.1におけるデータ転送ではDATA0とDATA1という二種類のパケットIDが用意されていて，これを交互に送ります．最初にDATA0というパケットIDを持つデータ・パケットを送ったら，次はDATA1というパケットIDを持つデータ・パケットを送るということを繰り返します．Sequenceビットは次のパケットIDを設定するビットです．ここで設定した値と実際にターゲットから到達したパケットIDが一致しないと

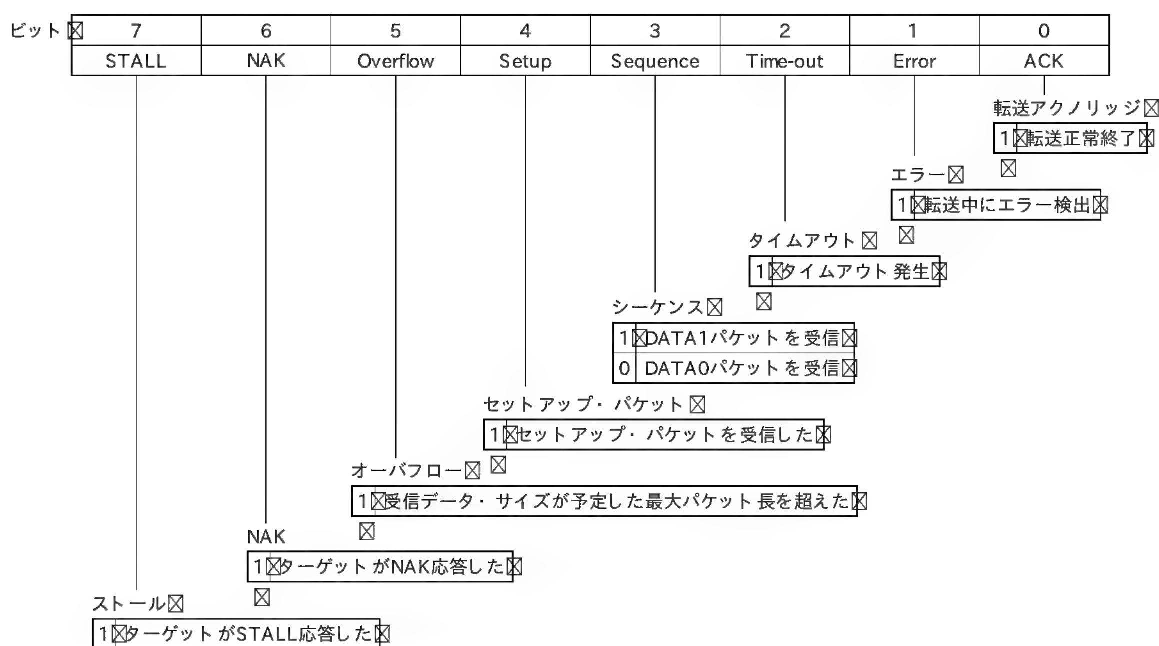


図4　USBステータスの構成

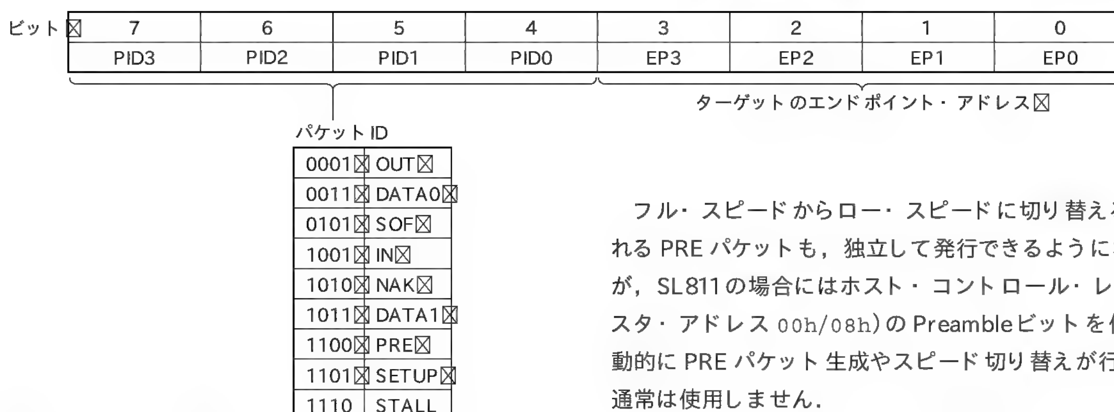


図5　パケットID/デバイス・エンドポイントの構成

パケットが受信されません．

Sequnceビットが‘0’ならばDATA0，‘1’ならばDATA1パケットということになります．

▶ Setup

SETUPパケット受信時に‘1’になります．SETUPパケットを発行するのはホストのみであり，今回はホスト動作を行わせるので，このビットは使用しません．

▶ Overflow

パケット受信時に用意したバッファ以上のデータが送られてくると‘1’になります．

▶ NAK

送ったパケットに対して相手がNAKで応答するとこのビットが‘1’になります．NAKは相手側のバッファがまだ使用中であったり（OUT方向），送るべきデータがまだ用意できていない（IN方向）の場合に発生するものなので，NAKが戻ってきたらホストはリトライを行います．

▶ STALL

送ったパケットに対して相手がSTALLで応答するとこのビットが‘1’になります．STALLはサポートされていないデバイス・リクエストを行ったなど，相手がエラーを検出したことを示すものです．STALLを受け取った場合，ホストは転送動作をエラー終了させます．

● パケットID/デバイス・エンドポイント（03h/0Bh）

ライトのみ有効です．リードすると，USBステータス・レジスタがアクセスされます．ビット配置は図5のように，上位4ビットがUSBバス上に送出したいパケットID，下位4ビットがターゲットのエンドポイント・アドレスの指定になります．ターゲット・デバイスのUSBバス上のアドレスはUSBアドレス・レジスタ（04h/0Ch）で指定します．

パケットIDはUSBバスの上に出るときはPIDに設定した値とそれを反転した値がペアになって8ビット・データとして送出されます．SL811の場合，PIDは任意の値がセットできるので，デバッグなどで利用するのには便利かもしれません．

フル・スピードからロー・スピードに切り替えるときに使われるPREパケットも，独立して発行できるようになっていますが，SL811の場合にはホスト・コントロール・レジスタ（レジスタ・アドレス00h/08h）のPreambleビットを使うことで自動的にPREパケット生成やスピード切り替えが行われるので，通常は使用しません．

● USBホスト転送カウント（04h/0Ch）

リードのみ有効な8ビットのレジスタです．ホスト・ベース・レングス・レジスタに設定した値と，実際に転送されたデータ・サイズの差がセットされます．IN方向の伝送時，USBステータスでACKビットが立って正常終了したにもかかわらずこの値がゼロになっていない場合は，ターゲットが要求サイズ以下のデータしか送ってこなかったということを示しています．

● USBアドレス（04h/0Ch）

ライトのみ有効なレジスタです．リードするとUSBホスト転送カウント・レジスタへのアクセスになります．アクセスしたいターゲット・デバイスのUSBアドレスをセットします．USBアドレスは7ビット長なので最上位ビットはつねに‘0’です．接続された直後のデバイスのアドレスは00hになっています．

実際にターゲットと接続した場合には，適宜別のアドレスをSET_ADDRESSリクエストを使って指定します．今回のサンプルでは02hを指定しています．

● コントロール・レジスタ1（05h）

レジスタのビット配置は図6のようになっています．電源投入後，すべてのビットは‘0’にクリアされます．

▶ SOF ena/dis

SOFパケットの自動生成を行うか，マニュアルで生成するかを指定します．‘1’をセットすると自動生成になります．USBの規格上ではSOFは必ず1ms周期で発生させなくてはならないので，通常は自動生成機能を使います．自動生成機能を使った場合，SOF Low Counter（0Ehライト）とSOF High Counter（0Fhのビット[5:0]）で生成周期を指定します．

▶ J-K State Force，USB Engine Reset

2ビットの設定値によってUSBバス・ラインの状態を設定できます．通常動作中はどちらのビットもゼロ（00b）にしておきます．01bならばUSBバスはリセット状態（D＋，D－とも“L”レベル）にします．

J-K State Forceビットが‘1’のときはD＋ラインがUSB Engine Resetビットの設定値に（‘1’を設定すると“H”レベル），D－ラインが設定値とは逆の状態になります．

なお，USBの場合フル・スピード（12Mbps）とロー・スピード（1.5Mbps）ではJステート，Kステートの状態が反対になり

P
1
2
3
4
5
6
A1
A2

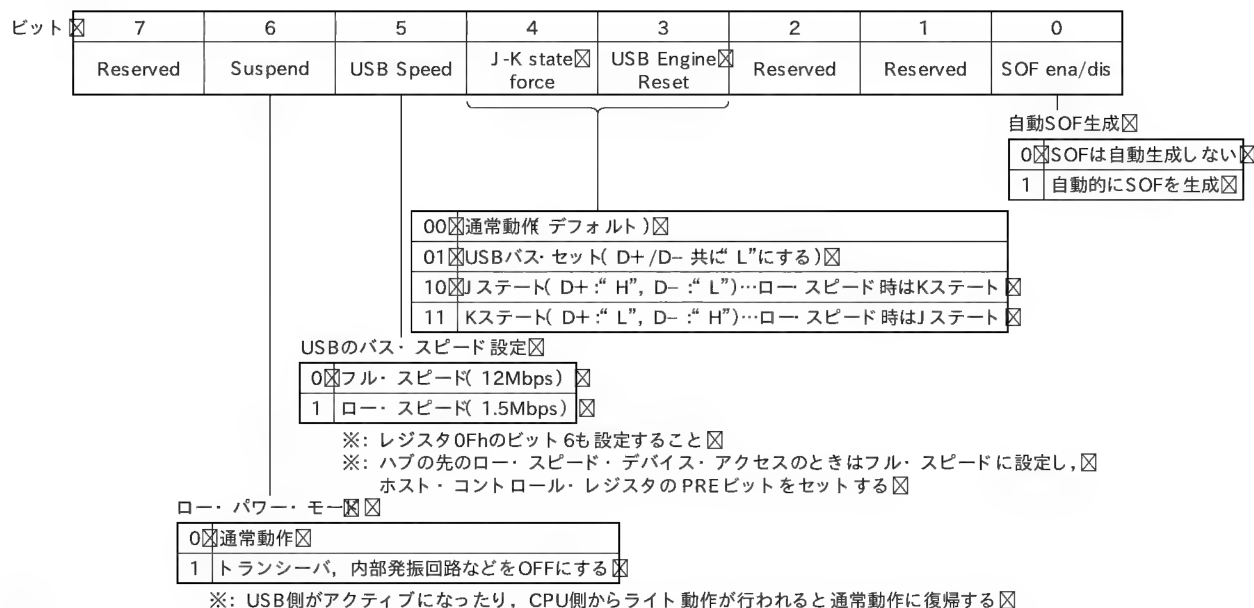


図6 コントロール・レジスタ1の構成

ます．たとえば，D＋が“H”レベル，D−が“L”レベルの状態はフル・スピードではJステートですが，ロー・スピードではKステートです．

▶ USB Speed

USBのバス・スピードを設定します．‘0’でフル・スピード，‘1’でロー・スピードになります．通常は‘0’で使用しますが，SL811に直接ロー・スピード・デバイスを接続したときは，このビットを‘1’にしてロー・スピード動作に切り替えます．

このビットを‘1’にしてロー・スピードで使う場合，レジスタ 0Fh のビット6のData Polarity Swapビットも‘1’にしてデータ極性を反転させる必要があります．

ハブの先に接続されたロー・スピード・デバイスにアクセスする場合，フル・スピード・モードのままで切り替えは不要(USB Speedビット，Data Polarity Swapビットとも‘0’)です．アクセスする前にホスト・コントロール・レジスタ(00h/08h)のビット7(Preambleビット)を‘1’にセットしてロー・スピード・デバイスへのアクセスであることをハブに通知するようにします．

▶ Suspend

SL811をサスペンド(ロー・パワー)状態にします．USBトランシーバの電源を切り，内部RAMをサスペンド状態にし，内部クロックも停止させてデバイスの消費電流を抑えます．USBバスに動きがあると通常の動作に復帰します．CPU側からサスペンド状態を解除したい場合にはSL811に対してデータ・ライトを行います．

● 割り込みイネーブル・レジスタ(06h)

SL811からCPUに対する割り込み発生条件を指定するレジスタです．該当する割り込み要因が成立するとCPUに割り込みがかかるとともに割り込みステータス・レジスタ(0Dh)のビットがセットされます．

ビット配置は図7のようになっています．今回のサンプルでは割り込みは使用しませんが，割り込みステータスをポーリングして利用するため，このレジスタを使用しています．

▶ USB-A，USB-B

SL811には転送要求のためのレジスタが2組あり，それぞれUSB-A，USB-Bと名付けられています．USB-A側が転送動作中でもUSB-B側に次の要求をセットすることができるため，転送完了の割り込みも別々に用意されています．

USB-A側(レジスタ 00h～)にセットした転送要求が完了したときに割り込みを発生させる場合にはUSB-Aビットを‘1’に，USB-B側(レジスタ 08h～)の転送が完了したときに割り込みを発生させるにはUSB-Bビットを‘1’にします．

▶ SOF Timer

SOFの自動生成を行ったとき，このビットを‘1’にしておくと，SOFに同期して割り込みが発生します．SOF自動生成はSOF Low Counter(レジスタ・アドレス 0Eh)，SOF High Counter(レジスタ・アドレス 0Fh のビット[6:0])を設定後，コントロール・レジスタ(レジスタ・アドレス 05h)のSOF ena/dis(ビット0)をセットします．

▶ Inserted/Removed

スレーブ(USBターゲット)デバイスが挿抜されたときに‘1’になります．

▶ Device Detect/Resume

コントロール・レジスタ1(レジスタ・アドレス 05h)のSuspendビット(ビット6)の設定で意味が変わります．

通常動作時(Suspend＝‘0’)にこのビットが‘1’になっていると，ターゲット・デバイスの接続が検出されたときに割り込

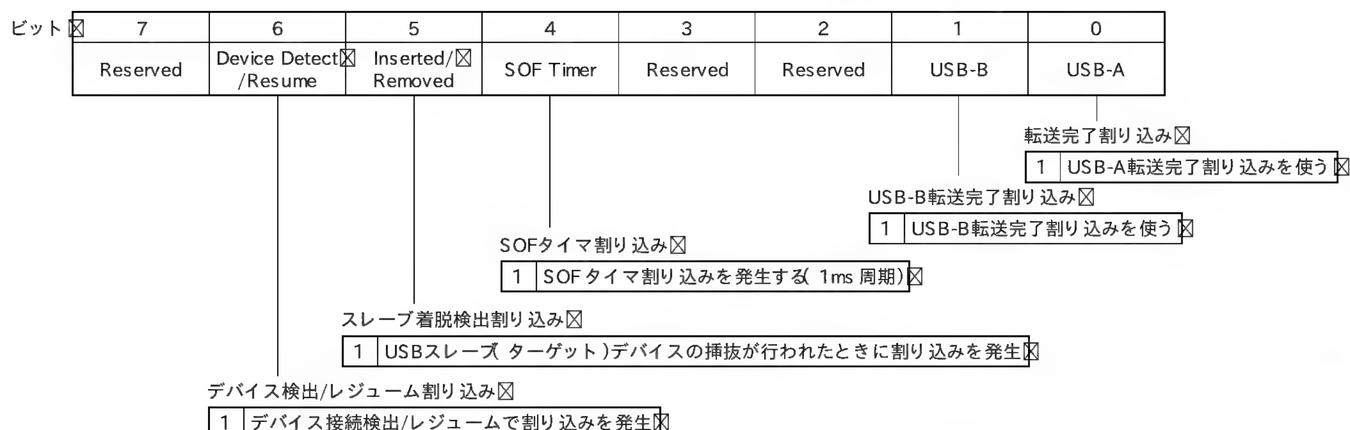


図7 割り込みイネーブル・レジスタの構成

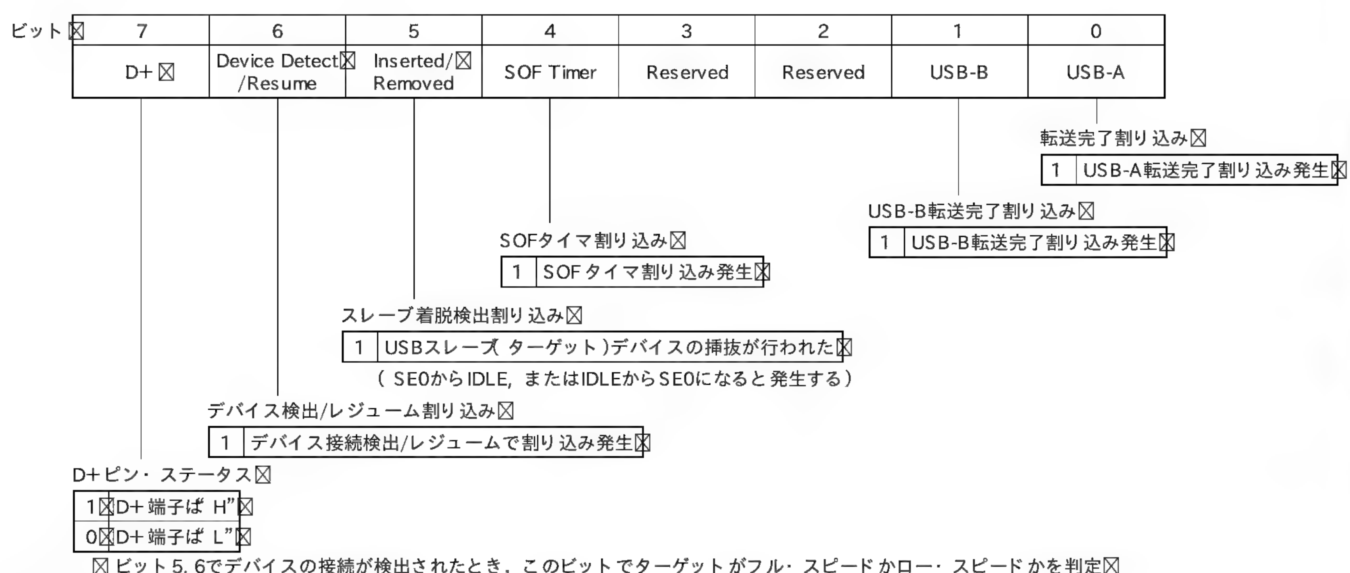


図8 割り込みステータス・レジスタの構成

みが発生します．

サスペンド時(Suspend＝'1')のときにこのビットが'1'になっていると，サスペンドから復旧したとき(レジューム)に割り込みが発生します．

● USBアドレス・レジスタ(07h)

リードのみ有効です．書き込みは行わないでください．USBターゲット・デバイスとして動作した場合に，ホスト側から与えられたUSBバス・アドレスがセットされます．USBバス・アドレスは7ビットなので，最上位ビットは無効です．今回はホスト動作なので，このレジスタは使用しません．

● 割り込みステータス・レジスタ(08h)

ビット[6:0]は割り込みイネーブル・レジスタと同様です(図8)．イネーブル・レジスタで設定した割り込み条件が成立すると割り込みが発生するとともに，割り込みステータス・レジスタの該当ビットが'1'になります．

ビット7(D＋ピン・ステータス)は割り込みではなく，D＋ラインの状態が読み出されます．ビット5, 6によってデバイスの接続が検出された場合，D＋が'1'ならばフル・スピード・デバイス，'0'ならばロー・スピード・デバイスが接続されたことがわかります．各ビットは'1'が書き込まれると'0'にクリアされます．

● ハードウェア・レビジョン(0Eh)

リードのみ有効です．図9のように上位4ビットでデバイスの種別やシリコン・レビジョンを表します．

● SOFカウンタLow(0Eh)

ライトのみ有効で，8ビットすべてが有効です．リードするとハードウェア・レビジョン・レジスタが読めてしまうので注意が必要です．

SL811にはSOFパケットを自動的に一定周期で送出する機能がありますが，この送出周期を14ビットのカウンタで決定しています．SOFカウンタLowレジスタには14ビットのカウント値の下位8ビットを設定し，上位6ビットはSOFカウンタ

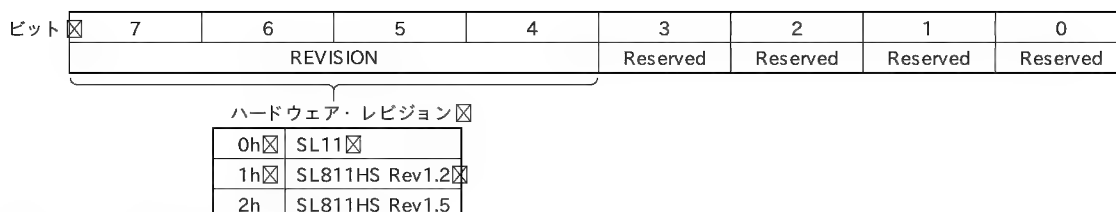


図9　ハードウェア・レビジョンの構成

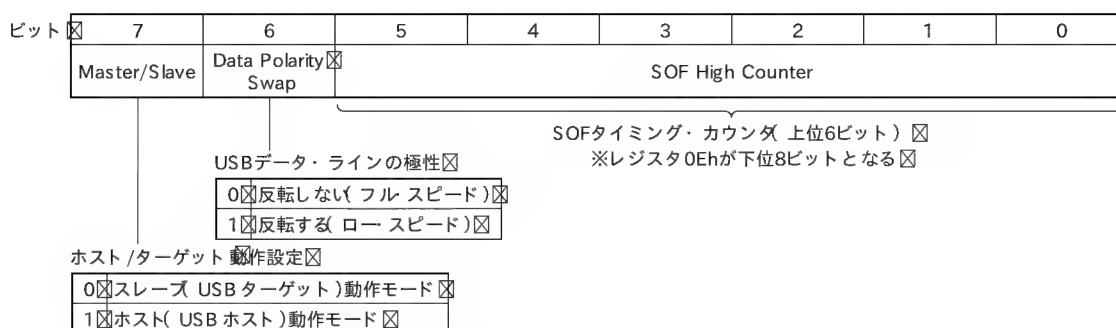


図10　SOFカウンタHigh/コントロール・レジスタ2の構成

High/コントロール・レジスタ2(レジスタ・アドレス0Fh)に設定します．

カウンタのクロック・ソースは12MHzなので，たとえばUSB規格どおりの1ms周期でSOFパケットを自動送出したい場合には設定値は2EE0H(=12000)になります．したがって，SOFカウンタLowレジスタには下位8ビットのE0を，SOFカウンタHigh/コントロール・レジスタ2の下位6ビットには2Ehをセットすることになります．

● SOFカウンタHigh/コントロール・レジスタ2(0Fh)

ライトのみ有効です．ビット配置は図10のようになっています．

▶ SOF High Counter

SOFパケットの自動生成機能を使った場合に，SOFカウンタLow(レジスタ・アドレス0Eh)に下位8ビット，SOF High Counterに上位6ビットをセットします．

なお，このレジスタに書き込みを行ったとき，SOFカウンタは自動的にクリアされます．

▶ Data Polarity Swap

フル・スピードで接続した場合には'0'，ロー・スピード時には'1'にします．USBの場合データ転送はNRZIエンコーディングしていますが，アイドル時の状態がフル・スピードとロー・スピードでは逆になっているため，このビットを使ってロー・スピードのときに反転させるというわけです．

このビットを'1'にするのはSL811に直接ロー・スピード・デバイスをつないだときだけです．ハブの先にロー・スピード・デバイスがつながっているときには，このビットは'0'のままにしておきます．

▶ Master/Slave

SL811はUSBのホストとターゲットのいずれにもなることができます．このビットはSL811をホスト・モードで動作させるか，ターゲット・モードで動作させるかを決めるものです．'1'をセットするとホスト・モード，'0'ならばターゲット・モードです．今回はホスト・モードで動作させるので，'1'にしています．

● SOFカウント値(0Fh)

リードのみ有効で，14ビットのSOFカウンタの上位8ビットが読み出されます．SOFカウンタはSOFカウンタ値としてセットした値からダウン・カウントされていき，0になるとリロードされるとともにSOFが発行されます．これを利用すると，SOFカウント値を見れば次にSOFが発行されるまでのおおよその時間がわかります．SOFカウンタのクロックは12MHzなので，このレジスタのカウント値の1カウントは(1/12MHz)×64=5.33μsとなります．

4 SL811の基本的な制御方法

実際にSL811にターゲットを接続する前に，SL811を使ったホスト・コントローラの操作方法について，簡単にまとめておきましょう．

起動後のSL811に対する操作は次のようになります．

(1) SL811の初期化

SL811の初期化は図11のような手順で行います．もともとレジスタも少なくシンプルなデバイスなので，初期化もそれほど難しくはありません．

▶ 内部レジスタ類の初期化

マスタ・モードの設定，SOFの周期設定などを行った後，割り込みイネーブル・レジスタ(レジスタ・アドレス`06h`)で必要な割り込みをイネーブルにします．今回のサンプルではD＋ビット，Reservedのビット以外はすべて'1'にして割り込み要因はすべて許可していますが，実際には割り込みとしては受けず，割り込みステータスをポーリングして処理しています．

▶ USBバスのリセット

SL811の初期化が終わった後，コントロール・レジスタ(レジスタ・アドレス`05h`)のビット[4:3]を利用してUSBバスを10ms以上リセット状態(D＋/D－ともに"L"レベル)にし，その後解除します．

▶ デバイスのスピード検出

SL811の下につながっているデバイスがあるか，また，つながっているならばそれがフル・スピード・デバイスなのかロー・スピード・デバイスなのかを判定し，コントロール・レジスタ(レジスタ・アドレス`05h`)，およびSOFカウンタHigh/コントロール・レジスタ2(レジスタ・アドレス`0Fh`)を設定します．

(2) パケットの送出

SL811の場合には非常に基本的な動作しかサポートしていません．逆にいえば単にパケットを発行したいという場合にはあまり余計な設定にわずらわされることもなく簡単に済ませられます．パケットの発行は図12のように行います．

▶ ターゲットのUSBアドレスを指定

ターゲット・デバイスのUSBバス・アドレスをUSBアドレス・レジスタ(レジスタ・アドレス`04h/0Ch`)にセットします．リセット直後はUSBターゲット・デバイスのUSBバス・アドレスは`00h`になっているので，最初のパケット発行時は`00h`を指定することになるでしょう．

▶ パケットIDとエンドポイント・アドレスの指定

送出したいパケットのパケットIDとエンドポイント・アドレスをUSBパケットID，デバイス・エンドポイント・レジスタ(レジスタ・アドレス`03h`)に設定します．

▶ バッファへのデータ・セット(OUT方向の場合)

SETUPパケットやデータ・パケットでOUT方向(ターゲットにデータを送る方向)の場合，送りたいデータの中身をSL811の内部バッファにセットします．内部バッファはレジスタ・アドレス`10h`から`FFh`の間です．

SL811にはレジスタ・アドレスの自動インクリメント機能があり，先頭アドレスを指定すればあとはデータだけ書き込めば自動的にアドレスが更新されるので，データ・セット時には便利でしょう．

▶ 転送データ長のセット

転送したいデータ長をホスト・ベース・レングス・レジスタ(レジスタ・アドレス`02h/0Ah`)に設定します．

▶ 転送データ領域の先頭アドレスのセット

OUT方向の場合には先に書き込んだデータ領域の先頭を，

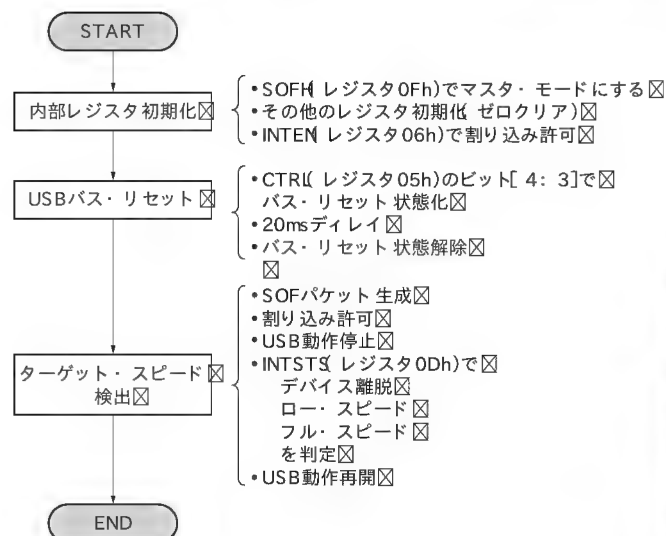


図11 SL811の初期化手順

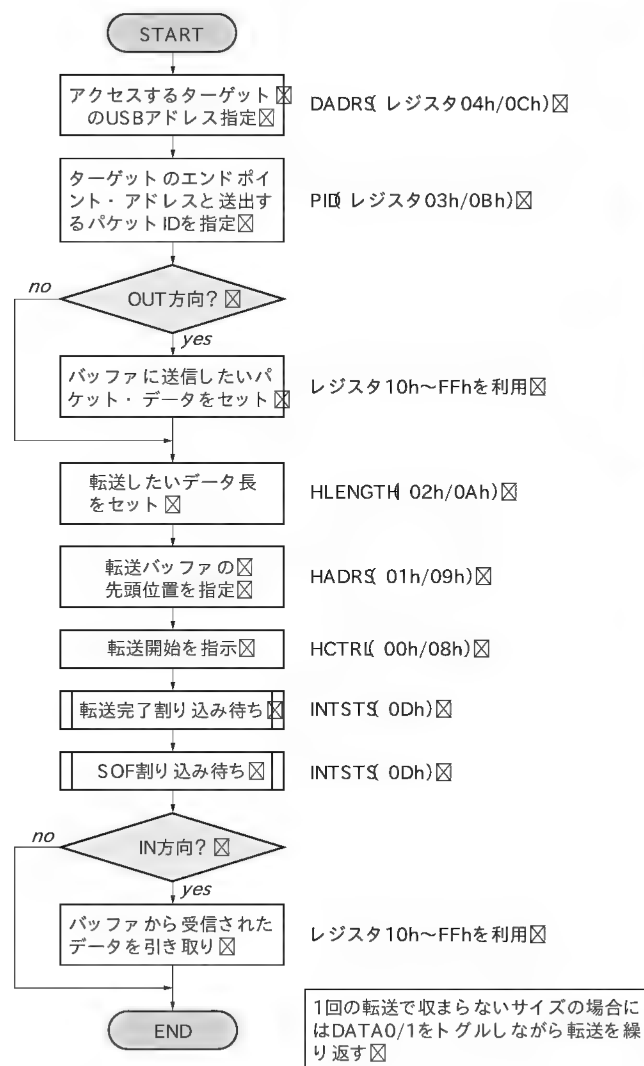


図12 パケットの送出手順

IN方向の場合にはターゲットから受け取ったデータを格納するバッファ領域の先頭アドレスを，ホスト・ベース・アドレス・レジスタ(レジスタ・アドレス 01h/09h)に設定します．

▶ 転送開始

ホスト・コントロール・レジスタ(レジスタ・アドレス 00h/08h)のEnableビット(ビット1)とArmビット(ビット0)を'1'にして転送開始を指示します．

このとき，データがOUT方向ならDirectionビット(ビット2)を'1'，アイソクロナス(CRCを無視する)ならISOビット(ビット4)を'1'，DATA1パケットならDataToggleビットを'1'にします．

SOFビットはどちらでもかまいませんが，アイソクロナス伝送でもない限り，SOFを待つ必要はあまりないので，通常は'0'で良いでしょう．

また，Preambleビット(ビット7)は，SL811の先にハブが接続されているとき，このハブの先につながれたロー・スピード・デバイスをアクセスするためのものなので，今回のサンプルのようにSL811とターゲット・デバイスを直結している場合にはセットしません．

▶ 転送完了待ちと完了ステータス取得

転送動作完了割り込みの発生を待ちます．今回はUSB-A側だけ使っているので，割り込みステータス・レジスタ(レジスタ・アドレス 0Dh)のUSB-Aビット(ビット0)が'1'になるのを待ちます．

割り込みが発生したら，USBステータス・レジスタ(レジスタ・アドレス 03h/0Bh)をチェックして転送動作が正常に終了したかどうかを確認します．

▶ データの取得(IN方向)

IN方向の場合，転送完了にともなって，受信されたデータがバッファに格納されるので，これを取得します．設定したバッファ・サイズよりもターゲットから送られてきたデータ・サイズが少なかった場合には，転送動作としては正常終了となりますが，USBホスト転送カウント・レジスタ(レジスタ・アドレス 04h/0Ch)が0になっていないことから，途中で転送が打ち切られたことがわかります．

データ・サイズが1パケットに入りきらない場合には，DATA0/1をトグルしながら次のデータ転送を行います．

5 USBターゲットの基本的な初期化手順

パケットの発行の方法が分かったところで，ではどのような順序でUSBターゲット・デバイスにアクセスしていったらよいのかを見ていきましょう．

USBの場合，コントロール・エンドポイント(エンドポイント・アドレス0)に対してデバイス・リクエストと呼ばれる8バイトのコマンドを送ります．デバイス・リクエストのフォーマットはUSBの規格で決まっていて，デバイスを認識したり，基本的なコントロールを行うようなものについては規格化されています．

デバイス・リクエストのフォーマットについてはUSBの規格書や書籍などを読んでもらうとして，それらをどのように使えば良いのかを示したのが図13です．

USBターゲット・デバイスは，接続された直後やUSBバス・リセットがかかった後はアドレス0になっているので，最初はターゲット・アドレスを0としてアクセスします．

接続が確認されたら，アドレス0のデバイスにGET_DESCRIPTOR要求でデバイス・ディスクリプタを指定したデバイス・リクエストを発行して，まずベンダIDやプロダクトID，コントロール・エンドポイント(エンドポイント0)のサイズなどを取得します．

さらに細かい情報を得るためにコンフィグレーション・ディスクリプタを，やはりGET_DESCIPTORを使って取得します．気をつけなくてはならないのは，コンフィグレーション・ディスクリプタを指定したときにはコンフィグレーション・ディスクリプタだけではなく，従属するインターフェース・ディスクリプタやエンドポイント・ディスクリプタなども送られてくるということです．つまり，コンフィグレーション・ディスクリプタに対する応答データ・サイズは，USBターゲットごとに異

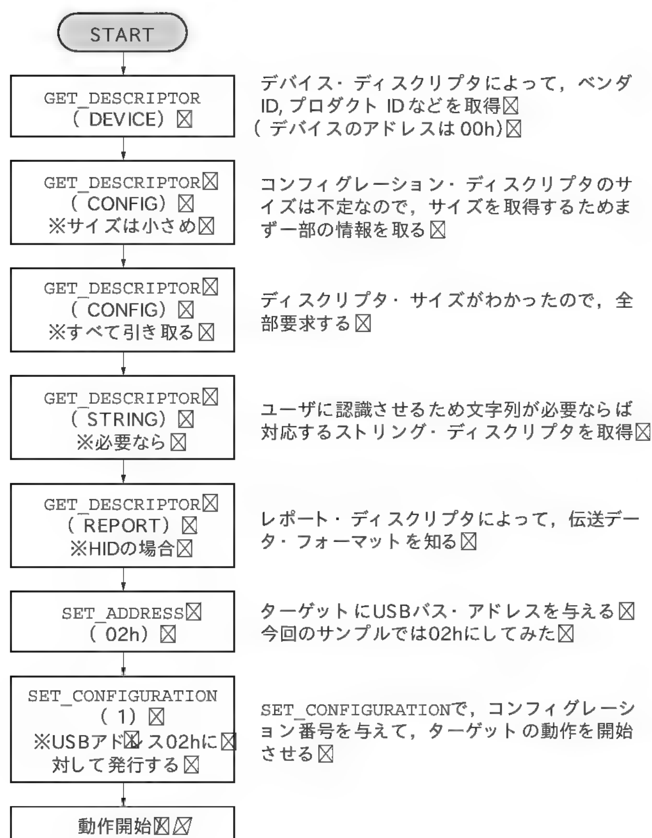
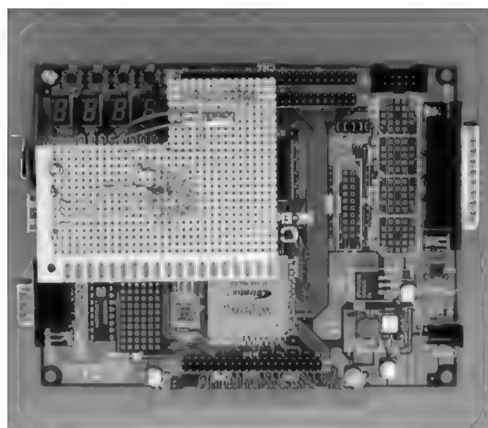


図13 USBターゲットの初期化手順



（a）拡張基板を実装したStratix評価キット



（b）拡張基板

写真1　試作したSL811搭載PCI評価ボード



写真2　SL811搭載CompactFlashカード型USBホスト・カード

なります．

そこで，まずホストはコンフィグレーション・ディスクリプタの先頭部分だけを取得します．コンフィグレーション・ディスクリプタの2バイト目と3バイト目が引き連れている全ディスクリプタのサイズの合計になっているので，この値を使って再度GET_DESCRIPTORで要求をかけることにより，全ディスクリプタを取得します．

これでUSBターゲットの消費電流などもわかります．また，ターゲットの準拠しているクラスなどもわかるので，PnP対応のOSならばここで必要なドライバをロードしてくることになるでしょう．

動作させても問題がないと判断できたら，ターゲットにUSBバス・アドレスを与えます．今回のサンプルでは02hに決め打ちしています．これが完了すると，ターゲットは指定したアドレスに切り替わるので，ここから先はアドレス0ではなく，与えたアドレスを使ってアクセスすることになります．

続いてSET_CONFIUGRATIONによってUSBターゲットを動作開始させます．コンフィグレーション番号は1からで，複数のコンフィグレーションがあるときは1から始まるいずれかの動作モードを指定して動作を開始させせることになります．今回想定しているUSBターゲットであるキーボードは，コンフィグレーションは1しかないので，1を指定して動作を開始させています．

これで最低限の初期化は終わりです．あとはベンダ・リクエスト/クラス・リクエストやデータ・エンドポイントを使ってデータのやり取りを行うことになります．

6 SL811とUSBキーボードをつないでみよう

● SL811搭載USBホスト・カード

それでは実際にSL811を使ってターゲット・デバイスと通信してみることにしましょう．

まず，SL811を搭載したUSBホスト・カードを用意する必要があります．今回は28ピンPLCCのSL811を入手したので，これをPC/AT互換機用拡張ボードに実装してみます．拡張ボードとしてはISAバスが簡単なのですが，すでにISAバスを実装したPCが手元にないので，写真1に示すようにPCI評価ボードの上にSL811を実装し，PCIバスと8ビット・ローカル・バスをブリッジする回路をFPGAに実装することで，PC/AT互換機のI/O空間にSL811をマッピングするようにしました．

また，もっと手軽に動作させてみたいという場合には，写真2に示すCompactFlashカード型USBホスト・カードREX-CFU1（ラトックシステム）を使ってみるのもよいでしょう．PCカード・アダプタに差し込んでPCカードに変換し，ノート・パソコンなどのPCカード・スロットに差し込んでPCカード・コントローラを初期化すれば，CFカード内に実装されているSL811のレジスタが，PC/AT互換機のI/O空間にマッピングされます．

作成したサンプル・プログラムは，I/Oを直接叩いてブリッジの初期化を行ったり，SL811にアクセスするので，MS-DOS（Windows 98の起動ディスクなどでも可）から行いました．コンパイラはLSI-C86の試食版を使いました．

USBターゲットには，とりあえず手元にあった，TK-U12FVSV（ELECOM）というUSBキーボードを使用しました．

● 標準クラス準拠のデバイス

いきなり仕様も何もわからないようなものをどうやってつなぐのか，と考え込んでしまいそうになりますが，キーボードは標準化団体であるUSB Implementers Forum, Inc（http://www.usb.org/）で規定したHID（Human Interface Device）クラスに準拠して設計されています．ちなみに，HIDクラスには，ほかにマウスやジョイスティックなども含まれています．

実はHIDクラスの場合，データ・フォーマットなどもフレキシブルになっていて，どのようなフォーマットでデータを送るかということを，レポート・ディスクリプタと呼ばれるテーブ

ルで表すようになっています．このため，あらゆるHIDデバイスに対応するためにはこのレポート・ディスクリプタを解析して自分の欲しいデータが何バイト目の何ビット目のデータとなるのかなどを調べなくてはならず，少々めんどうなことになります．しかし，今回のようにつなぐ相手を決めて，そのデバイス専用に対応することにするならば，あらかじめ相手から送られてくるデータ・フォーマットを手作業で解析しておいて，送られてきたデータを処理するプログラムを決め打ちで書いてしまえばよいので，比較的簡単です．

クラスとして標準化されているものには，このほかにもオーディオ・デバイス（マイクやスピーカなど）やマスストレージ・デバイス（HDDやCD-ROM，USBメモリなど）など，いろいろなものがあります．Windowsでメーカ専用のドライバを組み込まなくても利用できるデバイスは，だいたいこれらのクラスに準拠していると思ってよいでしょう．これらの標準クラス準拠のデバイスであれば，SL811を使って直接コントロールできるでしょう．

● キーボードの素性

HIDデバイスはデータ転送用にインタラプトINエンドポイントを利用しますが，キーボードが持っているエンドポイントの数やエンドポイント・アドレスがわからないと，ホストはどのエンドポイントにアクセスすればよいのかがわかりません．USBの場合，このような情報のやり取りやデバイスに対するコマンド送出などはコントロール・エンドポイントを使って行います．コントロール・エンドポイントはすべてのUSBデバイスが必ず持っているもので，エンドポイント・アドレスは0です．

USBデバイスの場合，デバイスに関する情報はディスクリプタと呼ばれる情報テーブルに記述され，ホストはUSB標準リクエストの一つであるGET_DESCRIPTORリクエストを使ってこの情報を取得します．ディスクリプタとしてホストに伝える情報は多岐にわたるため，ディスクリプタ・テーブルも種類ごとに整理されています．

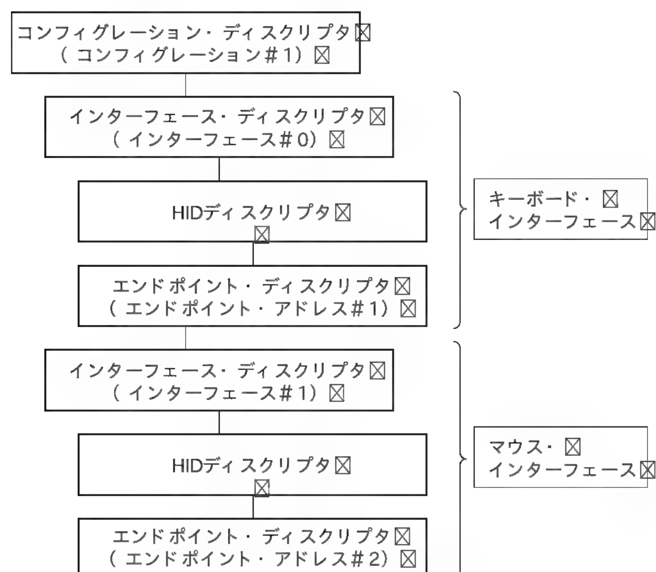


図14　ディスクリプタ・テーブルの並びと階層構造

USBデバイスの場合，一つのデバイスの中に複数の機能（インターフェースと呼ばれる）を持ったり，内部の構成そのものを切り替える（コンフィグレーションと呼ばれる）ようなことができるようになっています．また，それぞれのインターフェースとホストの間のデータ転送用のバッファ（エンドポイント）を持ちます．

イメージとしては一つのデバイスの中にモード切り替えによる複数のコンフィグレーションがあり，それぞれのコンフィグレーションの中には一つ，ないしは複数のインターフェースがあり，さらにそれぞれのインターフェースがデータ転送用のエンドポイントを持っているという構造を思い浮かべればよいでしょう．

これに対応して，ディスクリプタのほうもデバイス，コンフィグレーション，インターフェース，エンドポイントというそれぞれの階層ごとに決められたフォーマットのテーブルになっています．

GET_DESCRIPTORリクエストでコンフィグレーション・ディスクリプタを要求すると，従属するインターフェースやエンドポイント・ディスクリプタなどをまとめて取得できるのですが，この順序はディスクリプタの階層構造に準じています．今回接続したキーボードが返してくるコンフィグレーション・ディスクリプタは，図14のようにコンフィグレーション・ディスクリプタに続いて0番目のインターフェース・ディスクリプタ，その次にはそのインターフェースに属するHIDクラスの情報を示すHIDディスクリプタと，使用しているエンドポイントの情報を示すエンドポイント・ディスクリプタが続きます．続いて1番目のインターフェース（今回のキーボードではマウスのようだ）のインターフェース・ディスクリプタ，HIDディスクリプタ，エンドポイント・ディスクリプタが続きます．

また，これら以外に付加的な情報，たとえばメーカ名（ベンダ名）やデバイス名（製品名），標準クラスの場合のクラス固有の情報などもディスクリプタという扱いになっています．ホストはターゲットに対してそれぞれのディスクリプタ情報を要求し，得られた情報からターゲットの素性を知るわけです．

● キーボードの各ディスクリプタの構造

今回使用したキーボードが持つディスクリプタとその内容を整理すると次のようになります．

▶ デバイス・ディスクリプタ

デバイス全体にわたる情報が格納されています．ベンダID，プロダクトIDなどが入ります．表3にデバイス・ディスクリプタの構造とキーボードが返してきた値を示します．ELECOMのキーボードなのですが，デバイスIDを見ると`04B4h`です．これはサイプレスのIDです．

▶ コンフィグレーション・ディスクリプタ

コンフィグレーション・ディスクリプタの構造と今回のキーボードが返してきたコンフィグレーション・ディスクリプタの値は表4のようになっています．

コンフィグレーションというのはデバイスの動作モードのようなものです．たとえば，自動車ゲーム用のコントローラとして動くモードと飛行機用のコントローラとして動くとき，ペダルやスイッチなどの意味を変えるようなことが簡単にできるわけです．

コンフィグレーション・ディスクリプタの中にはこのコンフィグレーションのときに存在するインターフェースの数や，このコンフィグレーションのときの最大消費電流などがわかるようになっています．

今回のキーボードの場合，デバイス・ディスクリプタのbNumConfigurationsバイトが`01h`なので，持っているコンフィグレーションは一つだけです．

▶ インターフェース・ディスクリプタ

一つのまとまった機能ブロックのことをインターフェースと呼んでいます．たとえば，オーディオ出力付きのジョイスティックであれば，オーディオ出力が一つのインターフェース，ジョイスティック機能で一つのインターフェースとなり，合計二つのインターフェースを持つことになります．今回のキーボードはコンフィグレーション・ディスクリプタのbNumInterfaceバイトが`02h`になっているので，二つのインターフェースがあることがわかります．

インターフェース・ディスクリプタの構造と，今回使用したキーボードが返してきた値は表5のようになっています．値の左側がインターフェース# 0，右側がインターフェース# 1です．bInterfceClassとbInterfaceSubClassのコードによって判定すると，# 0がキーボードで# 1がマウスということになり

表3 デバイス・ディスクリプタの構造とキーボードが返してきた値

オフセット	名 称	データ長	内 容	キーボードが返した値	意 味
＋0	bLength	1	ディスクリプタ長	12h	
＋1	bDescriptorType	1	デバイス・ディスクリプタ(01hに固定)	01h	
＋2	bcdUSB	2	USB準拠規格をBCDで表す(2.1なら0210)	0110h	1.1準拠
＋4	bDeviceClass	1	準拠しているUSBクラス(インターフェース・ディスクリプタで定義するなら00h)	00h	インターフェースで指定
＋5	bDeviceSubClass	1	サブクラス・コード(各クラスのドキュメント参照)	00h	インターフェースで指定
＋6	bDeviceProtocol	1	プロトコル・コード(各クラスのドキュメント参照)	00h	インターフェースで指定
＋7	bMaxPacketSize	1	EP0の最大パケット・サイズ(ロー・スピードなら8)	08h	
＋8	idVendor	2	ベンダIDコード	04B4h	サイプレス
＋10	idProduct	2	プロダクトID	0101h	0101h
＋12	bcdDevice	2	デバイス・リリース番号をBCDで表す	0001h	Version 1
＋14	iManufacture	1	製造メーカ名を表すストリング・ディスクリプタの番号	01h	01番
＋15	iProduct	1	製品名を表すストリング・ディスクリプタの番号	02h	02番
＋16	iSerialNumber	1	製品のシリアル番号を表すストリング・ディスクリプタの番号	00h	なし
＋17	bNumConfiguration	1	設定可能なコンフィグレーションの数	01h	一つのみ

表4 コンフィグレーション・ディスクリプタの構造とキーボードが返してきた値

オフセット	名 称	データ長	内 容	キーボードが返した値	意 味
＋0	bLength	1	ディスクリプタ長	09h	
＋1	bDescriptorType	1	コンフィグレーション・ディスクリプタのID(02h)	02h	
＋2	wTotalLength	2	このコンフィグレーションに従属する各ディスクリプタの総サイズ	003B	59バイト
＋4	bNumInterface	1	このコンフィグレーションが持っているインターフェース数	02h	二つ(マウスとキーボード)
＋5	bConfigValue	1	コンフィグレーション番号(SET_CONFIGURATION使用する)	01h	
＋6	iConfiguration	1	このコンフィグレーションを表すストリング・ディスクリプタの番号	04h	
＋7	bmAttribute	1	ビット7：予約 ビット6：'1'：セルフパワー，'0'：バス・パワー ビット5：'1'：リモート・ウェイクアップ機能あり ビット4： 0：予約	A0h	バス・パワー・デバイス，リモート・ウェイクアップ機能あり
＋8	bMaxPower	1	消費電流(2mA単位)	32h	32H(＝50)×2＝100mA

表5 インターフェース・ディスクリプタの構造とキーボードが返してきた値

オフセット	名称	データ長	内容	キーボードが返した値		意味	備考
				IF#0	IF#1		
+0	bLength	1	ディスクリプタ長	09h	09h		
+1	bDescriptorType	1	インターフェース・ディスクリプタのID(04h)	04h	04h		
+2	bInterfaceNumber	1	インターフェース番号(0から)	00h	01h		
+3	bAlternateSetting	1	オルタネート・セッティング選択のための値	00h	00h		
+4	bNumEndpoint	1	このインターフェースが持っているエンドポイント数	01h	01h		
+5	bInterfaceClass	1	このインターフェースが準拠しているUSB-IF定義クラス	03h	03h	HIDデバイス・クラス	HID 4.1章参照
+6	bInterfaceSubClass	1	USB-IF定義のクラス準拠の場合のサブクラス番号	01h	01h	ブート・デバイス(KB/MOUSE)	HID 4.2章参照
+7	bInterfaceProtocol	1	プロトコル・コード(クラスごとに定義される)	01h	02h	01:キーボード,02:マウス	HID 4.3章参照
+8	iInterface	1	このインターフェースを表すストリング・ディスクリプタの番号	05h	06h		

表6 HIDディスクリプタの構造とキーボードが返してきた値

オフセット	名称	データ長	内容	キーボードが返した値		意味	備考
				IF#0	IF#1		
+0	bLength	1	ディスクリプタ長	09h	09h		
+1	bDescriptorType	1	HIDディスクリプタのID(21h)	21h	21h	HIDディスクリプタは21h	HID 7.1章
+2	bcdHID	2	準拠しているHIDクラスのレビジョンのBCD表記	0100h	0100h	01.00準拠	
+4	bCountryCode	1	ハードウェアのターゲット国コード	00h	00h	指定なし	HID5.2.1章
+5	bNumDescriptor	1	このHIDに従属するクラス定義ディスクリプタの数	01h	01h	1個(レポート・ディスクリプタのみ)	HID5.2.1章
+6	bDescriptorType	1	従属しているクラス定義ディスクリプタの種類(1番目)	22h	22h	レポート・ディスクリプタ	HID7.1章
+7	bDescriptorLength	2	従属しているクラス定義ディスクリプタのサイズ(1番目)	41h	32h	IF#0: 65バイト IF#1: 50バイト	
+9	bDescriptorType	1	従属しているクラス定義ディスクリプタの種類(2番目)				
+10	bDescriptorLength	2	従属しているクラス定義ディスクリプタのサイズ(2番目)				
:	:	:	:				

表7 エンドポイント・ディスクリプタの構造とキーボードが返してきた値

オフセット	名称	データ長	内容	キーボードが返した値		意味	備考
				IF#0	IF#1		
+0	bLength	1	ディスクリプタ長	07h	07h		
+1	bDescriptorType	1	エンドポイント・ディスクリプタのID(05h)	05h	05h		USB2.0 9.4章
+2	bEndpointAddress	1	エンドポイント・アドレス	01h	02h	INエンドポイント・アドレス 01hと02h	
+3	bmAttribute	1	ビット[7:6]:予約済み('0'固定) ビット[5:4]: UsageType (00:データ・エンドポイント) ビット[3:2]: SynchronizationType (00: No Synchronization) ビット[1:0]:エンドポイント種別 (11:インタラプト)	03h	03h	インタラプト・エンドポイント	USB2.0 9.6.6章
+4	wMaxPacketSize	2	このエンドポイントで1回に遅れる最大サイズ	08h	05h		
+6	bInterval	1	エンドポイント・アクセス周期	18h	30h	IF#0: 24ms IF#1: 48ms周期	

ます．

インターフェース・ディスクリプタでは，その機能ブロックがどのようなものなのかを記述しています．標準クラスに準拠しているならば，そのクラス・コードが入ります．

今回使用したキーボードの場合には，キーボード単体なのですがディスクリプタ上ではマウスも入っていて，マウス付きキーボードという扱いになっています．このため，インターフェース・ディスクリプタも二つあります．両方ともHIDクラスでProtocol Codeフィールドの値により，片方がキーボード，もう一方がマウスであることが示されています．ただし，後で紹介するレポート・ディスクリプタの解析結果からは，このマウス・インターフェース側にはマウスの位置情報などの実体がなく，マウスとしては機能していないようです．

▶ HIDディスクリプタ

今回使用したキーボードはHIDデバイス・クラスに準拠しているので，HIDディスクリプタを持っています．HIDディスクリプタは準拠しているHIDクラスのバージョンや従属するレポート・ディスクリプタ(実際のデータ伝送フォーマットを決定する)などが記述されます．

HIDディスクリプタの構造とキーボードが返した値をまとめたのが表6です．レポート・ディスクリプタとは，これがホストとの間でやりとりするデータの中身のフォーマットを決めるためのディスクリプタのことです．

▶ エンドポイント・ディスクリプタ

各インターフェースが使用しているエンドポイントの種別やサイズ，方向(インなのかアウトなのか)や，エンドポイント・アドレスなどの情報が入ります．今回使用したキーボードはマウス付きキーボードという扱いになっていて，それぞれのインターフェースごとに一つずつインタラプト・イン・エンドポイントを持っています．エンドポイント・ディスクリプタの構造とキーボードが返した値は表7のようになっています．

wMaxPacketSizeがキーボード用のほうは8バイトですが，マウス用のほうが5バイトというのが少し変わっているかもしれません．bIntervalも24msと48msというぐあいで，半端な値になっています．

▶ ストリング・ディスクリプタ

中身はUNICODE文字列です．UNICODEというと難しそうですが，半角英数字ならば00hを付加するだけなので単純です．ストリング・ディスクリプタはこのUNICODE文字列によるデバイス名，メーカ名などを納めたものの先頭にサイズ情報とストリング・ディスクリプタであることを表すコード(03hに固定)を付けた構造になっています．ストリング・ディスクリプタは複数持つことができるようになっていて，番号で区別しています．

たとえば，デバイス・ディスクリプタの中のiManufactureには製造メーカ名を表すストリング・ディスクリプタの番号が記述されていて，この番号を使ってGET_DESCRIPTORでストリング・ディスクリプタを要求するとターゲットからはメーカ名文字列が返されるといわけです．

今回使用したキーボードではiManufactureが01h，iProductが02hになっていたので，1番目の文字列が製造メーカ名，2番目が製品名ということになります．そのほかキーボードが返してくるストリング・ディスクリプタを整理したのが表8です．メーカ名が堂々とCypressであったり，インターフェースの名称がEP1 interruptであったりと，ちょっと首を傾げたくなるような文字列が返ってきますが，動けばよいというところでしょう．

なお，ストリング・ディスクリプタ#3は欠番，#4はディスクリプタ上ではサイズが20hとなっていますが，実際には18hバイトしか返ってこないようでした．

なお，ストリング・ディスクリプタのインデックス0(#0)は文字列ではなく，言語コードを表す2バイトのLANG_IDデータが返されます．

▶ レポート・ディスクリプタ

レポート・ディスクリプタはHIDデバイスとの間でやり取りするデータのフォーマットを決めるものです．HIDデバイスはキーボード，ジョイスティック，マウスなどですが，ジョイスティック一つとってみてもジョイパッドや操縦桿型のものだけ

表8　キーボードが返してきたストリング・ディスクリプタ

ID	+0	+1	+2	+4	+6	+8	+10	+12	+14	+16	+18	+20	+22	+24	+26
0	04h	03h	0409h(English)												
1	12h	03h	“C”	“y”	“p”	“r”	“e”	“s”	“s”	“ ”					
			0043h	0079h	0070h	0072h	0065h	0073h	0073h	0020h					
2	1Ch	03h	“U”	“S”	“B”	“ ”	“K”	“e”	“y”	“b”	“o”	“a”	“r”	“d”	“ ”
			0055h	0053h	0042h	0020h	004Bh	0065h	0079h	0062h	006Fh	0061h	0072h	0064h	0020h
4	20h	03h	“H”	“I”	“D”	“ ”	“K”	“e”	“y”	“b”	“o”	“a”	“r”		
			0048h	0049h	0044h	0020h	004Bh	0065h	0079h	0062h	006Fh	0061h	0072h		
5	1Ch	03h	“E”	“P”	“1”	“ ”	“I”	“n”	“t”	“e”	“r”	“r”	“u”	“p”	“t”
			0045h	0050h	0031h	0020h	0049h	006Eh	0074h	0065h	0072h	0072h	0075h	0070h	0074h
6	1Ch	03h	“E”	“P”	“2”	“ ”	“I”	“n”	“t”	“e”	“r”	“r”	“u”	“p”	“t”
			0045h	0050h	0032h	0020h	0049h	006Eh	0074h	0065h	0072h	0072h	0075h	0070h	0074h

ではなく，ハンドル型になったり，付いているスイッチも本物に近づけようとすればするほど増えていき，複雑になります．

あまりにもバリエーションがありすぎてデータ・フォーマットを統一することができないため，レポート・ディスクリプタというものによって転送されるデータのフォーマットを記述して，ホストはそれに基づいてデータ長やどのビットが何の意味なのかを判定するわけです．

たとえば，デバイスがマウスで「Button0からButton2の範囲で値は0か1を取る1ビット・データ三つ分のボタン，その次の5ビットは未使用で，続く2バイトが8ビット長のX方向，Y方向の移動距離で値は−128から＋127の範囲をとる」というように記述すれば，これはホストに3バイトのデータを送り，下位の3ビット分がボタン，次の5ビットがダミーで，その次にX方向，Y方向の移動量が送られる3ボタン・マウスであるということになります．ちょっと考えればわかるとおり，この解析はかなりめんどうなので，今回は手作業解析にしました．

実際にキーボードが返してきたレポート・ディスクリプタ（インターフェース1のみ）を表9に示します．右のほうに対応するHIDやHUT（UsageTable）ドキュメントの章番号を入れておきました．これらのドキュメントはhttp://www.usb.org/からダウンロードできます．

暗号のような文字列ですが，UsagePage（Generic Desktop），Usage（Keyboard）が以下の情報がキーボード関係であることを示します．

Collecitor（Application）からEndCollectionの間でデータ・フォーマットが記述されます．さらにUsagePage（keyCode）に

表9　キーボードが返してきたレポート・ディスクリプタ（インターフェース1のみ）

データ	アイテム	種　別	意　味	参照ドキュメント
05h 01h	UsagePage（Generic Desktop）	グローバル・アイテム	Generic Desktop	HID6.2.2.7章　HUT3章
09h 06h	Usage（Keyboard）	ローカル・アイテム		HID6.2.2.8章　HUT4章
A1h 01h	Collection（Application）	メイン・アイテム	キーボード/マウス系	HID6.2.2.4章
05h 07h	UsagePage（Key Codes）	グローバル・アイテム	以下はキーコード	HID6.2.2.7章　HUT10章
19h E0h	Usage Minimum（E0h）	ローカル・アイテム	E0h：左Ctrlキー（LSB） E1h：左Shiftキー E2h：左Altキー E3h：左GUIキー E4h：右Ctrlキー E5h：右Shiftキー E6h：右Altキー	HID6.2.2.8章　HUT10章
29h E7h	Usage Maximum（E7h）	ローカル・アイテム	E7h：右GUIキー（MSB）	HID6.2.2.8章
15h 00h	Logical Minimum（0）	グローバル・アイテム	論理的な最小値0	HID6.2.2.7章
25h 01h	Logical Maximum（1）	グローバル・アイテム	論理的な最大値1	HID6.2.2.7章
75h 01h	Report Size（1）	グローバル・アイテム	1個あたり1ビット	HID6.2.2.7章
95h 08h	Report Count（8）	グローバル・アイテム	8個分LSBから並ぶ	HID6.2.2.7章
81h 02h	Input（Data Variable Absolute）	メイン・アイテム	データ入力	HID6.2.2.4章
95h 01h	Report Count（1）	グローバル・アイテム	1個分のデータ	HID6.2.2.7章
75h 08h	Report Size（8）	グローバル・アイテム	1個あたり8ビット	HID6.2.2.7章
81h 01h	Input（Constant）	メイン・アイテム	定数データ（未使用）	HID6.2.2.4章
95h 03h	Report Count（3）	グローバル・アイテム	3個分のデータ	HID6.2.2.7章
75h 01h	Report Size（1）	グローバル・アイテム	1個あたり1ビット	HID6.2.2.7
05h 08h	Usage Page（LED）	グローバル・アイテム	LEDへの出力データ	HID6.2.2.7章　HUT11章
19h 01h	Usage Minimum（01h）	ローカル・アイテム	01h：Num Lock（LSB） 02h：Caps Lock	HID6.2.2.8章　HUT11章
29h 03h	Usage Maximum（03h）	ローカル・アイテム	03h：Scroll Lock（MSB）	HID6.2.2.8章　HUT11章
91h 02h	Output（Data Variable Absolute）	メイン・アイテム	データ出力	HID6.2.2.4章
95h 05h	Report Count（5）	グローバル・アイテム	5個分並ぶ	HID6.2.2.7章
75h 01h	Report Size（1）	グローバル・アイテム	1個あたり1ビット	HID6.2.2.7章
91h 01h	Output（Constant）	メイン・アイテム	データ出力	HID6.2.2.4章
95h 06h	Report Count（6）	グローバル・アイテム	6個分並ぶ	HID6.2.2.7章
75h 08h	Report Size（8）	グローバル・アイテム	1個あたり8ビット（1バイト）	HID6.2.2.7章
15h 00h	Logical Minimum（00h）	グローバル・アイテム	論理的な最小値0	HID6.2.2.7章
26h FFh 00h	Logical Maximum（00FFh）	グローバル・アイテム	論理的な最大値255	HID6.2.2.7章
05h 07h	UsagePage（Key Codes）	グローバル・アイテム	以下はキーコード	HID6.2.2.7章　HUT10章
19h 00h	Usage Minimum（00h）	ローカル・アイテム		HID6.2.2.8章
2Ah FFh 00h	Usage Maximum（00FFh）	ローカル・アイテム		HID6.2.2.8章
81h 00h	Input（Data Array）	メイン・アイテム	データ入力	HID6.2.2.4章
C0h	End Collection	メイン・アイテム	終了	HID6.2.2.4章

よって，以下で扱う値が「キーコード」データであることを示します．

ここから先は文章だけ読んでいると混乱しそうになりますが，実際にキーボードとの間でやり取りされるデータ・フォーマット（表10）と見比べながら見ていくとわかりやすいでしょう．

UsageMinimum，UsageMaximumによって，データとして取り込まれるキーが`E0h`の左Ctrlキーから`E7h`の右GUI（Windowsキー）キーであることが示されます．つまり，シフトキーやCtrlキーを押したときのデータです．

続くLogical Minimum，Logical Maximumが，これらを表す値の最大値と最小値です．スイッチであり，2値しかとらないのでMinimumが0，Maximumが1となります．つまり，CtrlやALTなどのキーの状態がそれぞれが'1'が'0'で表現されるわけです．

続くReportSizeがデータ長，Report Countがデータの数になります．この場合には1と8なので，1ビット・データが8個，つまり1バイト分のデータとなることを示します．続くInputによって，このデータが実際に取得されるという扱いになります．つまり，最初の8ビット・データはビット0がキーコード`E0h`にあたる左Ctrlキー，ビット7が`E7h`にあたる右GUIキーになるわけです．

次はReportCountが1，ReportSizeが8でInputではConstantが指定されているので，これは8ビットの定数データ，つまり未使用領域です．したがって，2バイト目は「未使用」となります．

次にUsagePageが切り替えられてキーボードではなくLEDに移ってしまっていますが，こちらもチェックしておきましょう．UsageMinimumが`01h`，UsageMaximumが`03h`となっています．LEDのページで`01h`はNumLock，`02h`がCapsLock，`03h`がScrollLockのLEDになっています．続いてOutputがあるので，これらの3ビット・データが下位から順に該当するLEDの点滅データとして使われることを示します．つまり，ビット0がNumLock，ビット2がScrollLockのLEDにあたるわけです．

次にReportCountが5，ReportSizeが1となってOutput（Constant）なので，5ビット分のパディングが行われていること，つまり上位5ビットは使われていないことを示します．

次はReportCountが6，ReportSizeが8なので，8ビット・データが6個，そしてLogicalMinimum，LogicalMaximumのあとUsagePageがキーコードに切り替えられているので，`00h`～`FFh`の値すべてが扱われること，UsageMinimumとUsageMaximumでキー側もすべてのキーが対象となることを示してInputされています．

順序がややこしいのですが，要するにキーコードとして`00h`～`FFh`の間すべての値をとる，キーコードを表す8ビット・データが6個入力されるということです．

これによって，キーボードから入力されるデータが8バイト長であること，キーボードに送るデータが下位3ビットのみ有効であることが分かります．なお，キーボードからの入力はEP1（エンドポイント・アドレス1のエンドポイント）のインタラプト・イン・エンドポイントを使いますが，キーボードへの出力はHIDクラスのクラス・リクエストの一つであるSet Reportリクエストを使用します．

●レポート・ディスクリプタの作成ツール

レポート・ディスクリプタの構造解析はめんどうですが，レポート・ディスクリプタを作成するためのツールが下記のURLで公開されています．

`http://www.usb.org/developers/hidpage/`

サンプルのディスクリプタも用意されているので，これを使って実際にキーボードなどから返されたレポート・ディスクリプタを比較して，解析結果があっているかどうかの検証を行うのにも便利に利用できると思います．

表10　レポート・ディスクリプタが示すキーボードからの送信データ

バイト順	データ							
	ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0
+0	右GUI	右ALT	右SHIFT	右CTRL	左GUI	左ALT	左SHIFT	左CTRL
+1	未使用							
+2	キーコード1（最初に押されたキー）							
+3	キーコード2（キーコード1のキーを押しながらさらに押されたキー）							
+4	キーコード3（キーコード1，2のキーを押しながらさらに押されたキー）							
+5	キーコード4（キーコード1，2，3のキーを押しながらさらに押されたキー）							
+6	キーコード5（キーコード1，2，3，4のキーを押しながらさらに押されたキー）							
+7	キーコード6（キーコード1，2，3，4，5のキーを押しながらさらに押されたキー）							

（a）キーボードからの送信データ・フォーマット（データ・エンドポイントを使う）

バイト順	データ							
	ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0
+0						ScrollLock	CapsLock	NumLock

（b）キーボードへの送信（Set_Reportリクエストを使う）データ・フォーマット

COLUMN

Windowsの初期化手順

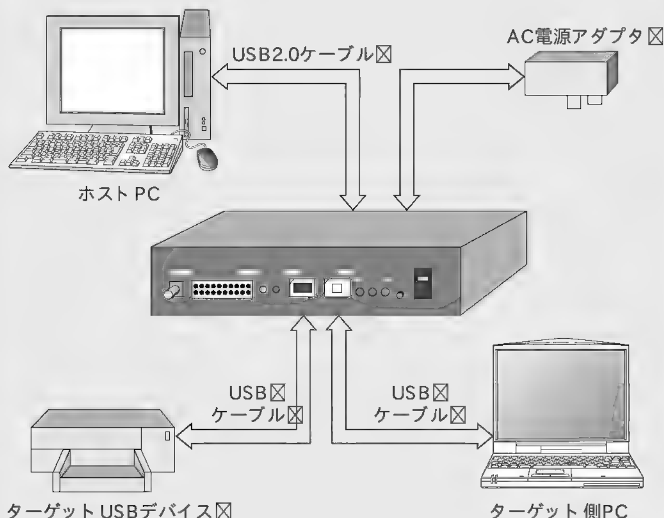
USB機器をWindowsマシンにつないだときには，一体どのようなリクエストが発行されるのでしょう．そこでWindowsマシンにUSB機器をつないだときの挙動を，USBバス・アナライザでトレースしてみました．今回使用したUSBバス・アナライザはハイ・スピードにも対応したBusCope(ヒロテック製)です．写真Aに外観を，図Aに接続図を，図BにGUIの画面例を示します．

このアナライザは，トレースしたデータをテキスト・ファイルで出力もできます．そのテキスト・ファイルの一部を切り出したものにコメントをつけたものをリストAに示します．

Windowsでは何に使うのかよくわからないほどストリング・ディスクリプタを繰り返し取りに来ます．ドライバなどが階層構造になっていて，それぞれの階層でデバイスを認識するたびに自分に必要な情報を取ってこようとするのでしょうか．Windows上ではこの後にドライバの組み込みなどが行われています．

はじめて接続したUSB機器が認識された後，一度USB機器を抜いて，再度同じポートに接続した場合は，最初のほうの挙動こそ同じですが，ドライバの登録は済んでいるので発行されるリクエストが途中で変わります．接続されたデバイスが過去に一度接続したことのあるデバイスの場合は，ストリング・ディスクリプタを何度も取得することはせずに，すぐにレポート・ディスクリプタの取得が行われるため，初期化時間が短くなっているようです．

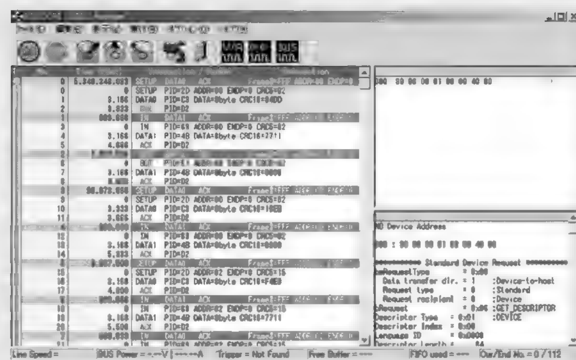
見えてしまえばどうということもない簡単な動きの連続ではありますが，見えるか見えないかというところには大きな違いがあるといえるでしょう．やはりこうして挙動がつかめるというのは，デバッグ時に威力を発揮します．USB機器の開発を行う場合には，やはり手元において置きたい道具でしょう．



図A　BusCopeの接続図



写真A　USBバス・アナライザBusCopeの外観



図B　BusCopeのGUI画面例

リストA　Windows2000にUSBキーボードをつないだときの動作

```
| No=[         0] Time=[ 5.348.348.083]nSec Transaction=[ SETUP  DATA0   ACK         ]
                                                          Frame#=FFF ADDR=00 ENDP=0
| No=[         0] Time=[             0]nSec Packet=[ SETUP  ] PID=2D ADDR=00 ENDP=0
| No=[         1] Time=[         3,166]nSec Packet=[ DATA0  ] PID=C3 DATA=8byte
| No=[         2] Time=[         3,833]nSec Packet=[  ACK   ] PID=D2
| 000: 80 06 00 01 00 00 40 00                        :......@.
| No=[         1] Time=[       973,833]nSec Transaction=[  IN    DATA1   ACK         ]
                                                          Frame#=FFF ADDR=00 ENDP=0
| No=[         3] Time=[             0]nSec Packet=[  IN    ] PID=69 ADDR=00 ENDP=0
| No=[         4] Time=[         3,166]nSec Packet=[ DATA1  ] PID=4B DATA=8byte
| No=[         5] Time=[         4,666]nSec Packet=[  ACK   ] PID=D2
| 000: 12 01 10 01 00 00 00 08                        :........
| No=[         2] Time=[     1,973,666]nSec Transaction=[  OUT   DATA1   ACK         ]
                                                          Frame#=FFF ADDR=00 ENDP=0
| No=[         6] Time=[             0]nSec Packet=[  OUT   ] PID=E1 ADDR=00 ENDP=0
| No=[         7] Time=[         3,166]nSec Packet=[ DATA1  ] PID=4B DATA=0byte
| No=[         8] Time=[         4,000]nSec Packet=[  ACK   ] PID=D2
| No=[         3] Time=[    96,947,833]nSec Transaction=[ SETUP  DATA0   ACK         ]
                                                          Frame#=FFF ADDR=00 ENDP=0
| No=[         9] Time=[             0]nSec Packet=[ SETUP  ] PID=2D ADDR=00 ENDP=0
| No=[        10] Time=[         3,333]nSec Packet=[ DATA0  ] PID=C3 DATA=8byte
```

GET_DESCRIPTOR(DEVICE)によってターゲット(キーボード)のデバイス・ディスクリプタを要求する．要求サイズは40h(64)バイト

ターゲットからデバイス・ディスクリプタが送られてくる．エンドポイント・サイズが8バイトなので転送は8バイトごとに分割して実行される．最初の要求なので，まず先頭8バイト分が送られてくる

ホスト側は続くデータの読み取りを行わず，ハンドシェーク・パケット(データがIN方向なので，逆向きのサイズ0バイトのOUTパケットを発行)を送り，終了させる

SET_ADDRESS(0x02)によって，アドレスを02hに設定することを要求

リスト A Windows2000 に USB キーボードをつないだときの動作(つづき)

```
| No=[        11] Time=[         3,666]nSec Packet=[  ACK   ] PID=D2
| 000: 00 05 02 00 00 00 00 00                        :........
| No=[         4] Time=[       973,833]nSec Transaction=[  IN    DATA1   ACK          ]
                                                         Frame#=FFF ADDR=00 ENDP=0
| No=[        12] Time=[             0]nSec Packet=[  IN    ] PID=69 ADDR=00 ENDP=0
| No=[        13] Time=[         3,166]nSec Packet=[ DATA1  ] PID=4B DATA=0byte
| No=[        14] Time=[         5,333]nSec Packet=[  ACK   ] PID=D2
| No=[         5] Time=[     8,971,666]nSec Transaction=[ SETUP  DATA0   ACK          ]
                                                         Frame#=FFF ADDR=02 ENDP=0
| No=[        15] Time=[             0]nSec Packet=[ SETUP  ] PID=2D ADDR=02 ENDP=0
| No=[        16] Time=[         3,166]nSec Packet=[ DATA0  ] PID=C3 DATA=8byte
| No=[        17] Time=[         4,000]nSec Packet=[  ACK   ] PID=D2
| 000: 80 06 00 01 00 00 12 00                        :........
| No=[         6] Time=[       973,833]nSec Transaction=[  IN    DATA1   ACK          ]
                                                         Frame#=FFF ADDR=02 ENDP=0
| No=[        18] Time=[             0]nSec Packet=[  IN    ] PID=69 ADDR=02 ENDP=0
| No=[        19] Time=[         3,166]nSec Packet=[ DATA1  ] PID=4B DATA=8byte
| No=[        20] Time=[         5,500]nSec Packet=[  ACK   ] PID=D2
| 000: 12 01 10 01 00 00 00 08                        :........
| No=[         7] Time=[       974,000]nSec Transaction=[  IN    DATA0   ACK          ]
                                                         Frame#=FFF ADDR=02 ENDP=0
| No=[        21] Time=[             0]nSec Packet=[  IN    ] PID=69 ADDR=02 ENDP=0
| No=[        22] Time=[         3,166]nSec Packet=[ DATA0  ] PID=C3 DATA=8byte
| No=[        23] Time=[         5,333]nSec Packet=[  ACK   ] PID=D2
| 000: B4 04 01 01 01 00 01 02                        :........
| No=[         8] Time=[       973,833]nSec Transaction=[  IN    DATA1   ACK          ]
                                                         Frame#=FFF ADDR=02 ENDP=0
| No=[        24] Time=[             0]nSec Packet=[  IN    ] PID=69 ADDR=02 ENDP=0
| No=[        25] Time=[         3,166]nSec Packet=[ DATA1  ] PID=4B DATA=2byte
| No=[        26] Time=[         6,000]nSec Packet=[  ACK   ] PID=D2
| 000: 00 01                                          :..
| No=[         9] Time=[       973,833]nSec Transaction=[  OUT   DATA1   ACK          ]
                                                         Frame#=FFF ADDR=02 ENDP=0
| No=[        27] Time=[             0]nSec Packet=[  OUT   ] PID=E1 ADDR=02 ENDP=0
| No=[        28] Time=[         3,333]nSec Packet=[ DATA1  ] PID=4B DATA=0byte
| No=[        29] Time=[         3,833]nSec Packet=[  ACK   ] PID=D2
| No=[        10] Time=[     1,973,500]nSec Transaction=[ SETUP  DATA0   ACK          ]
                                                         Frame#=FFF ADDR=02 ENDP=0
| No=[        30] Time=[             0]nSec Packet=[ SETUP  ] PID=2D ADDR=02 ENDP=0
| No=[        31] Time=[         3,333]nSec Packet=[ DATA0  ] PID=C3 DATA=8byte
| No=[        32] Time=[         3,833]nSec Packet=[  ACK   ] PID=D2
| 000: 80 06 00 02 00 00 09 00                        :........
| No=[        11] Time=[       973,833]nSec Transaction=[  IN    DATA1   ACK          ]
                                                         Frame#=FFF ADDR=02 ENDP=0
| No=[        33] Time=[             0]nSec Packet=[  IN    ] PID=69 ADDR=02 ENDP=0
| No=[        34] Time=[         3,166]nSec Packet=[ DATA1  ] PID=4B DATA=8byte
| No=[        35] Time=[         5,333]nSec Packet=[  ACK   ] PID=D2
| 000: 09 02 3B 00 02 01 04 A0                        :..;.....
| No=[        12] Time=[       973,833]nSec Transaction=[  IN    DATA0   ACK          ]
                                                         Frame#=FFF ADDR=02 ENDP=0
| No=[        36] Time=[             0]nSec Packet=[  IN    ] PID=69 ADDR=02 ENDP=0
| No=[        37] Time=[         3,333]nSec Packet=[ DATA0  ] PID=C3 DATA=1byte
| No=[        38] Time=[         5,166]nSec Packet=[  ACK   ] PID=D2
| 000: 32                                             :2
| No=[        13] Time=[       973,666]nSec Transaction=[  OUT   DATA1   ACK          ]
                                                         Frame#=FFF ADDR=02 ENDP=0
| No=[        39] Time=[             0]nSec Packet=[  OUT   ] PID=E1 ADDR=02 ENDP=0
| No=[        40] Time=[         3,333]nSec Packet=[ DATA1  ] PID=4B DATA=0byte
| No=[        41] Time=[         3,833]nSec Packet=[  ACK   ] PID=D2
| No=[        14] Time=[     1,973,500]nSec Transaction=[ SETUP  DATA0   ACK          ]
                                                         Frame#=FFF ADDR=02 ENDP=0
| No=[        42] Time=[             0]nSec Packet=[ SETUP  ] PID=2D ADDR=02 ENDP=0
| No=[        43] Time=[         3,333]nSec Packet=[ DATA0  ] PID=C3 DATA=8byte
| No=[        44] Time=[         4,333]nSec Packet=[  ACK   ] PID=D2
| 000: 80 06 00 02 00 00 FF 00                        :........
| No=[        15] Time=[       973,666]nSec Transaction=[  IN    DATA1   ACK          ]
                                                         Frame#=FFF ADDR=02 ENDP=0
| No=[        45] Time=[             0]nSec Packet=[  IN    ] PID=69 ADDR=02 ENDP=0
| No=[        46] Time=[         3,166]nSec Packet=[ DATA1  ] PID=4B DATA=8byte
| No=[        47] Time=[         4,833]nSec Packet=[  ACK   ] PID=D2
| 000: 09 02 3B 00 02 01 04 A0                        :..;.....
| No=[        16] Time=[       974,000]nSec Transaction=[  IN    DATA0   ACK          ]
                                                         Frame#=FFF ADDR=02 ENDP=0
| No=[        48] Time=[             0]nSec Packet=[  IN    ] PID=69 ADDR=02 ENDP=0
| No=[        49] Time=[         3,166]nSec Packet=[ DATA0  ] PID=C3 DATA=8byte
| No=[        50] Time=[         5,333]nSec Packet=[  ACK   ] PID=D2
| 000: 32 09 04 00 00 01 03 01                        :2.......

～以下略～
```

- SET_ADDRESSはデータ転送フェーズを伴わないので，IN方向の0バイト・パケットによって終了する．このINパケットに対してホストがACK応答し正常に終了した時点でデバイスのアドレスが切り替わる
- 再びGET_DESCRIPTOR(DEVICE)を発行してデバイス・ディスクリプタを要求する．サイズは12h(デバイス・ディスクリプタ・サイズ分)
- キーボードから先頭8バイト分のディスクリプタ情報がくる
- 今度は8バイト目で打ち切らず，続く8バイト・データも引き取る
- これでデバイス・ディスクリプタ全部(12h＝18バイト分)の転送終了
- サイズ0のOUTパケットで転送の正常終了を示す
- GET_DESCRIPTOR(CONFIGURATION)でコンフィグレーション・ディスクリプタを要求．サイズはとりあえず9バイト
- キーボードからコンフィグレーション・ディスクリプタの先頭8バイト分が送られてくる
- 続いて残りの1バイトも取得する
- 正常に受け取れたので0バイトのOUTパケットで終了する
- 再度GET_DESCRIPTOR(CONFIGURATION)でコンフィグレーション・ディスクリプタを要求．要求サイズはFFh(255)バイト
- 0～7バイト目のデータがくる
- 8～15バイト目のデータがくる

7 サンプル・プログラム

では，実際に作成したサンプル・プログラムについて説明します(本誌WebまたはInterGiga No.34に収録予定)．

サンプル・プログラムのソースはslkbd.cとhostctl.cの二つに分かれています．このうちSL811の操作を実際に行っているのがhostctl.cです．

hostctl()関数がエントリ部分で，slkbd.cのmain()から呼ばれます．hostctl()関数で行っていることは，先ほど説明した初期化手順そのものです．SL811の操作を簡単にするためにパケットの発行を行うusb_packet()関数を作り，これを使ってSETUPパケットの組み立てと発行を行うsetup_packet()，さらにこれを使ってデバイス・リクエストの一連の処理をまとめて行うdevice_request()関数を作成しました．

● usb_packet()

usb_packet()関数は，ターゲット・デバイスのバス・アドレス，エンドポイント・アドレス，パケット種別やデータ長，バッファの先頭アドレスなどの情報を受け取り，SL811の各レジスタを設定してパケット発行を行います．SL811の基本的な操作そのものなので，ここだけ切り出して流用することもできるでしょう．

● setup_packet()

SETUPパケットはエンドポイント0を使ったターゲットへのコマンドのようなものでデバイス・リクエストと呼ばれます．サイズはつねに8バイトです．setup_packet()はこのデバイス・リクエストの各フィールドごとの値を受け取って8バイトデータに組み立て直してusb_packet()を呼び出します．

● device_request()

デバイス・リクエストの処理(コントロール伝送)は，SETUPパケットの発行に続いて必要に応じてデータの入出力がともなわれ，最後に逆向き(データのIN方向ならOUTパケット，データOUT方向ならINパケット)でサイズ0のパケットを発行して完了します．

たとえばGET_DESCRIPTORのようなものなら，まずホストからSETUPパケットでGET_DESCRIPTORリクエストを構成している8バイトのコマンド・データを送ります．続いてホストがINパケットを発行すると，ターゲットがこれに応じてディスクリプタ情報を送ってきます．一度に送ることができるのはエンドポイント・サイズまでなので，データ数が多いときは繰り返しIN要求を行ってターゲットからデータを引き取ります．

最後にサイズ0のOUTパケットをターゲットに対して送出して一連の処理が終了します．

● サンプルの処理と実行

device_request()を使ってデバイス・ディスクリプタを取得し，続いてSET_ADDRESSをしてみました．もちろん，各種のディスクリプタ情報を取得した後で行ってもかまいません．このあとコンフィグレーション・ディスクリプタ，ストリング・ディスクリプタ，レポート・ディスクリプタなどを取得してみました．すでに整理したとおり，とてもおもしろい結果が得られました．続いてSET_CONFIGURAIONで動作開始を指示し，データ・リードに移ります．データ・リード中にPCの'Q'キーを押すとプログラムの実行を終了します．

実行結果(log.txt)では，起動後，次のように操作したログです．

1) 左のCTRL，SHIFT，ALT，GUI(Windowsキー)の順にキーを離さず，押していく(最後は左のCTRL＋ALT＋GUIが押された状態になる)
2) 手を離す
3) 'A'，'B'，'C'，'D'の順にキーを離さずに押していく(最後はABCDの四つを押している状態になる)
4) 手を離す
5) 'A'を押して離す
6) 'B'を押して離す
7) 'C'を押して離す
8) 'D'を押して離す
9) 'A'，'B'，'C'の順にキーを離さずに押していく(最後はABCの三つを押している状態になる)
10) 'A'，'B'，'C'の順にキーを離していく

結果を見ると，ほぼ予想どおりになっていることがわかります．キーデータの領域が6バイトもありますが，複数のキーが押されたときに押された順番に格納されていき，どのキーが同時に押されているのかがわかるようになっています．また，3)のステップのときにDキーを押したとたん，キーデータがすべて01hになってしまいました．多重押しでキーの正常な判定ができなくなったことを示しているのでしょう．

最初のCTRLキーなどの押下で，先頭バイトがビット単位で各キーに対応していることや，並びも確かにレポート・ディスクリプタの解析どおりであることもわかります．

まとめ

今回，SL811を使って実際にパケットの発行を行い，キーボードとの間のデータ伝送を行ってみました．SL811の仕様はコンパクトで簡易的なものですが，この大きさ，このピン数でUSBのホストとして利用できることは大きな利点でしょう．今まで自作のUSB機器を作成してもそれをほかの自作機器などからコントロールすることはできませんでした．また，マウスやキーボードといった汎用的な機器を自作機器の下につないで利用することも簡単にはいきませんでしたが，SL811を使うことでこれらの道が開けます．ホスト，ターゲットの両方ともPCから離れることで，USBを汎用の高速通信路として使う道も開けることでしょう．

くわの・まさひこ　パステルマジック

組み込み機器向けで480Mbpsにも対応したホスト/ターゲット・コントローラ

4 ハイ・スピード対応ホスト・コントローラ M66596の概要

加藤 智之/家田 淳/平野 実秋

第3章で取り上げたSL811同様，M66596もホストとターゲットの両方の機能を内蔵しているが，ここではホスト機能に注目して解説する．とくにM66596は，480Mbpsのハイ・スピードにも対応したホスト・コントローラなので，組み込み機器にストレージ機器を接続するような用途に最適である．（編集部）

はじめに

● USBの普及

近年，プラグ＆プレイの手軽さから，PC周辺機器にUSBインターフェースは欠かせないものになりました．特にプリンタ，ディジタル・スチル・カメラ，ディジタル・ビデオ・カメラ，マスストレージ機器，およびスキャナなどの高速，大容量のデータ転送が必要な製品は，USB2.0ハイ・スピード（480Mbps）へと移行しています．

一方で，USBインターフェースは従来のPC周辺機器だけではなく，ポータブル・オーディオ機器など，民生機器へも搭載されるようになってきました．

このような状況で，PCを経由せずにUSB機器どうしを接続し通信させたいという要求が高まっています．すでにプリンタとディジタル・スチル・カメラをUSBで接続し，直接印刷できるダイレクト・プリンティングが実現されていますが，それだけでなく音楽やビデオなどの大容量データを転送するという用途は確実に増えています（図1）．

また，USBは信号線が2本（電源を除く）しかなく，かつ高速であることから，図2のストレージ・メディアの例のように，機器内通信にUSBを使用するケースもあります．

このようにPCを経由せずにUSB機器どうしを接続する場合，必ずどちらかのデバイスがPCの代わりにホストの役割を担う必要があります．

● 組み込みシステムでの問題点

組み込みシステムの多くは，CPUの処理性能やメモリ空間などに大きな制限があります．このため，USBホストを実現する場合，PC向けのホスト・コントローラをそのままシステムに持ち込むのは難しい場合があります．

ここでは，組み込み機器向けに開発されたルネサステクノロジ製ハイ・スピード対応USBホスト/ペリフェラル・コントローラM66596を用いたUSBホストについて解説します．

1 M66596の特徴

● 汎用バスで容易に接続可

M66596は，ホスト・コントローラとして動作する場合でも，ペリフェラル・コントローラと同様の制御でデータ転送を実現できるのが特徴です．しかもハイ・スピード転送（480Mbps）に対応し，高速なデータ転送が可能です．

マイコンとは汎用的バス・インターフェースで接続できるので，さまざまなシステムに容易に組み込むことができます．ま

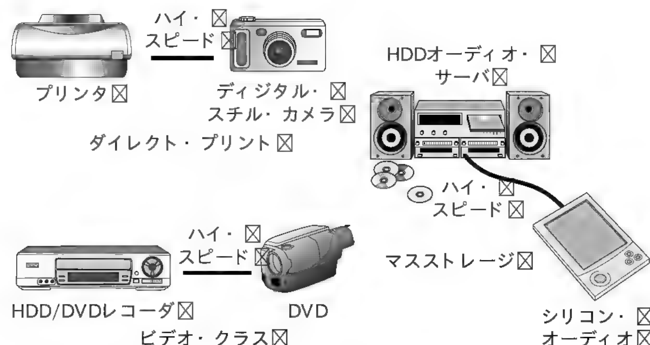


図1 USB機器同士で通信する応用例

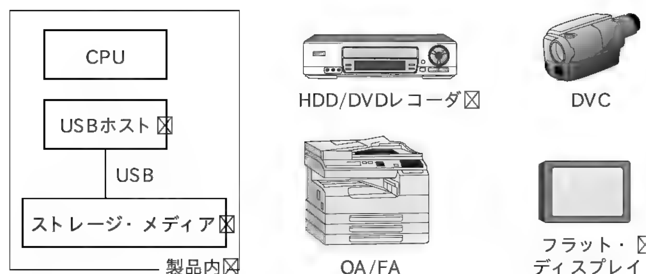


図2 機器の内部でUSB接続する応用例

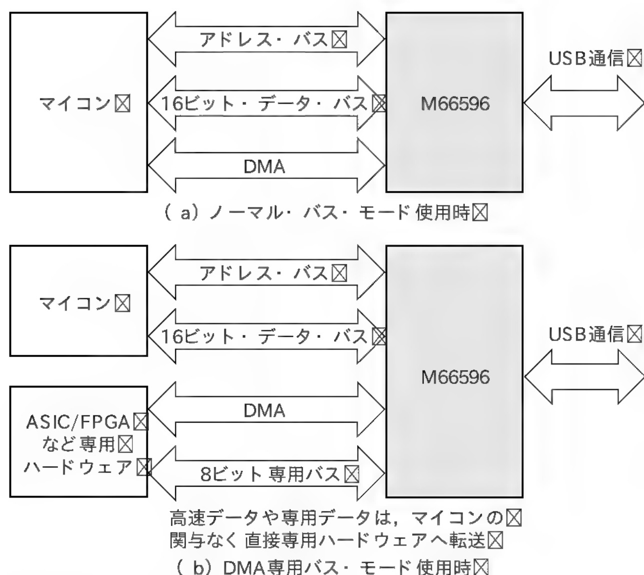


図3　M66596のバス・インターフェース仕様

表1　M66596のおもな仕様

USB機能	ホストまたはペリフェラル・モードをレジスタで設定可能
転送モード	ハイ・スピード・モード(480Mbps)，フル・スピード・モード(12Mbps) USBリセット時にハードウェアにより自動認識
転送タイプ	コントロール/バルク/アイソクロナス/インタラプト
通信パイプ(エンドポイント)	計8本 ●コントロール転送用1本 ●アイソクロナス/バルク転送兼用2本 ●バルク転送用3本 ●インタラプト転送用2本
ホスト・モード時の転送スケジューリング	ハードウェアによりSOFパケット，トランザクションの発行
内蔵バッファ	5Kバイト．使用する通信パイプに対してソフトウェアによりサイズを設定可能
バス・インターフェース	●汎用バス(セパレート・バス，マルチプレクス・バスを選択可能) ●独立した8ビット・バスもDMAで使用可能
DMA	2チャネル
パッケージ	64ピンLQFP，64ピンFBGA

た，CPUバス・インターフェースとは独立したDMA専用バス・インターフェースを装備し，高速大容量データをマイコンのバスを占有せずに転送することができます．

M66596のバス・インターフェース仕様を図3に示します．また M66596のおもな仕様を表1に示します．

● USBホストとして必要な機能

一般的にペリフェラル・コントローラと比べて，USBホストには次のような機能が必要になります．

- ペリフェラル機器のアタッチ/デタッチ(接続/切断)の検出
- 接続されたペリフェラル機器のステート管理
  USBリセット/サスペンド/レジュームおよびコントロール転送によるデバイス・ステート管理
- フレームの生成とパケットのスケジューリング
  μSOFまたはSOFパケットを生成し，要求されたトランザクションをUSBのプロトコルに従ってスケジューリングする
- データ転送
  ペリフェラル機器のエンドポイントに対してデータの転送を行う
- 転送エラーの検出
  破損パケットの受信，不正なPID，無応答などを検出

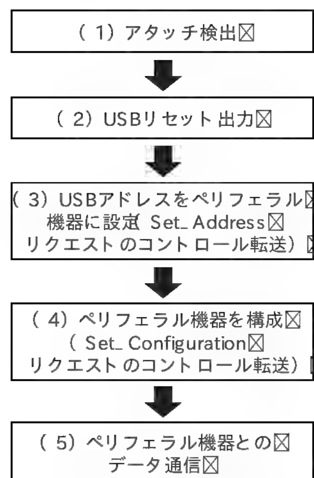


図4　エニュメレーションの流れ

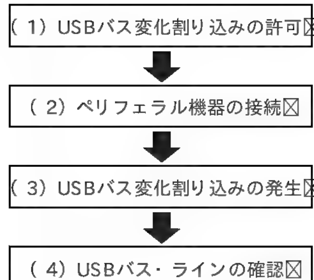


図5　アタッチ検出の流れ

2 USBペリフェラル機器の検出からデータ転送まで

次にペリフェラル機器のアタッチを検出してデータ転送を行うまでの，M66596の動作と制御方法について説明します．

● ケーブル接続からSOFパケットの出力まで

USBケーブルを接続された後，ホストは図4に示すエニュメレーション処理を行い，ペリフェラル機器との通信を開始します．

まず，ペリフェラル機器の接続を検出してからSOFパケットを出力するまでのM66596の制御方法について説明します．

● アタッチの検出

図5にアタッチ検出の流れを示します．ペリフェラル機器の接続は，USBバス(D＋，D－)の変化で検出します．USBホストは，D＋，D－を15kΩの抵抗でプルダウンするため，切断状態ではD＋，D－は両方とも“L”レベル(SE0ステート)となっています．

一方，ペリフェラル機器はケーブルが接続されると，フル・スピード/ハイ・スピード・デバイスはD+を，ロー・スピード・デバイスはD－を1.5kΩでプルアップします．このプルアップにより，USBバスはJステートの状態となります．

M66596はUSBバスの変化を検出すると割り込みを発生させ

る機能があります．マイコンはこの割り込みによってペリフェラル機器の接続を知ることができます．また，レジスタを参照することにより，D+，D−の状態がわかります．これにより接続されたのがハイ・スピード，フル・スピードの機器なのか，ロー・スピードの機器なのかを確認することができます．

● USBリセットの出力

ホスト・コントローラは，ペリフェラル機器の接続を検出すると，USBリセットを出力しペリフェラル機器のUSB機能をリセットします．USBリセットは，ホスト側がSE0を出力することで行いますが，このときリセット・ハンドシェーク・プロトコルと呼ばれる信号のやり取りを行い，通信速度を決定します．

M66596は，リセット・ハンドシェークを自動的に実行します．このためソフトウェアは，

(1) USBリセット出力を設定
(2) 時間待ち
(3) USBリセット解除
(4) Start Of Frame(SOF)パケットの出力設定

を行うだけで，接続されたペリフェラル機器の速度で，転送が可能な状態になります．図6にUSBリセットの処理の流れを示します．

また(3)のUSBリセット出力停止時にM66596のレジスタを参照すると，リセット・ハンドシェークの結果を確認することができます．図7にハイ・スピード・ペリフェラル機器が接続されてからSOFパケットを出力するまでのM66596の動作と制御方法をUSBバスの状態とともに示します．

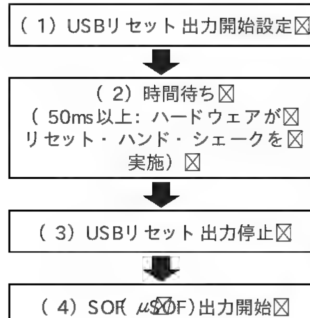


図6 USBリセットの処理の流れ

USBリセット出力停止後は，コントロール転送によりエニュメレーションの続きを行います．

3 M66596によるコントロール転送

次にエニュメレーション時に実行する，コントロール転送の制御方法について説明します．

USBのコントロール転送は，SETUPステージ，データ・ステージ，ステータス・ステージの三つのステージからなります．

M66596でコントロール転送を行う場合の，それぞれのステージでの制御方法について説明します．図8にコントロール・リード転送のシーケンスを示します．

● SETUPステージ

SETUPステージでは，コントロール転送のリクエストとそのパラメータを送信します．M66596は，送信データを設定す

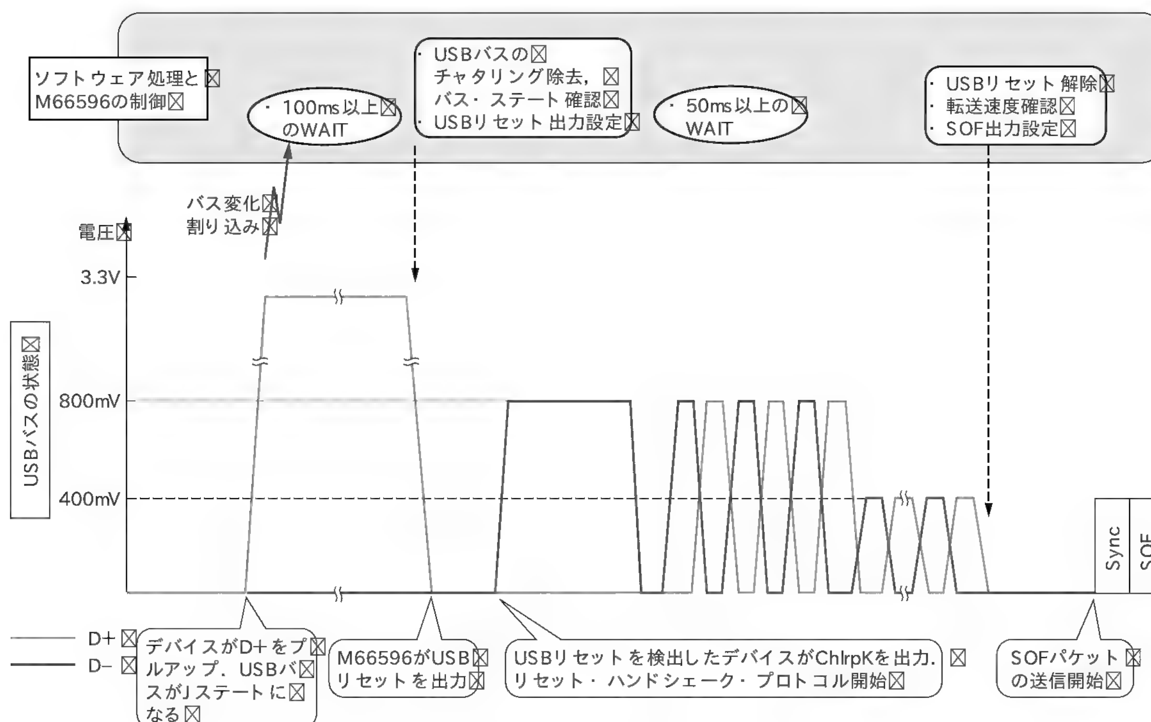


図7 ハイ・スピード・デバイスが接続された場合のアタッチからSOFパケット送信開始までの処理

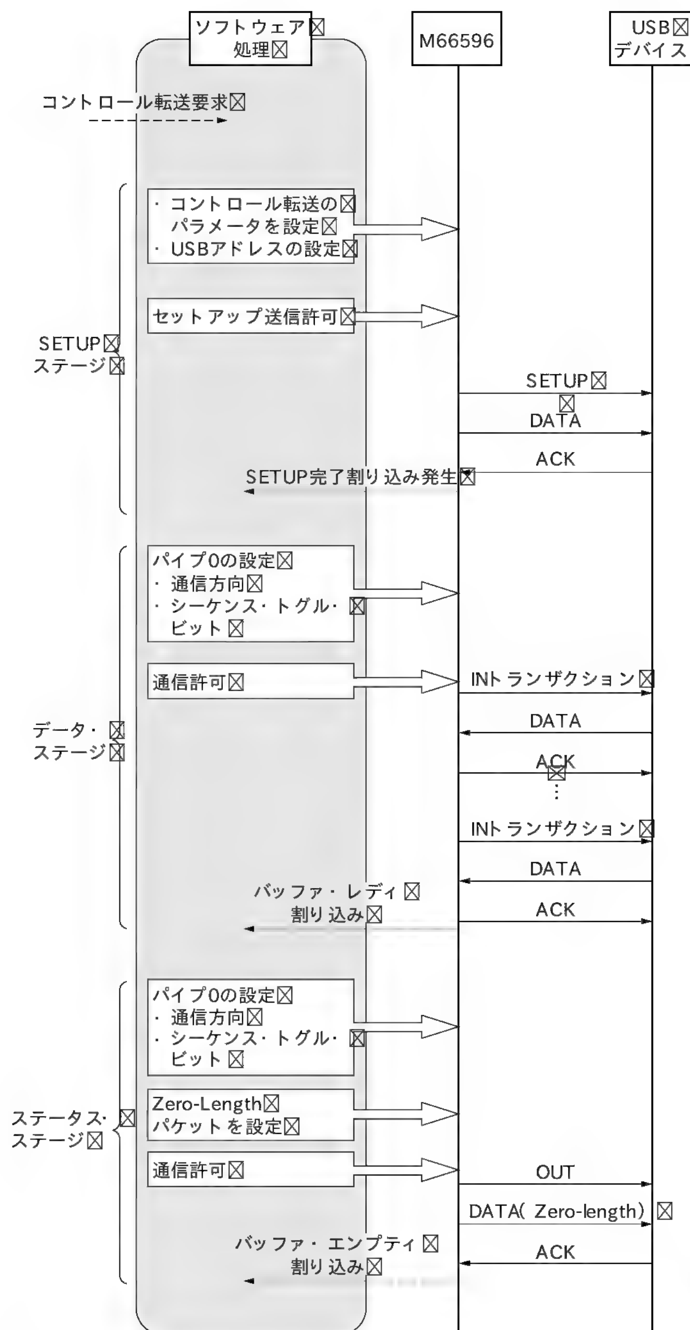


図8　コントロール・リード転送のシーケンス

るためのレジスタを持っています．ソフトウェアでレジスタにリクエスト・データを書き込み，送信を許可することにより，SETUPパケットとデータ・パケットを送信します．

M66596はペリフェラル機器からのACKパケットを受信すると，SETUP完了割り込みを発生させてソフトウェアにトランザクションの終了を通知します．

● データ・ステージ

送信したリクエストの内容に従い，データの送信または受信を行います．データ転送の方向，シーケンス・トグル・ビットの設定を行い，通信を許可すると，次の条件でM66596はトランザクションを発行します．

- IN方向（コントロール・リード）
  受信バッファに空きがある場合
- OUT方向（コントロール・ライト）
  送信データがバッファに書かれている場合

M66596は，256バイトのバッファをコントロール転送用に割り当てています．256バイトまでの転送であれば，複数のパケットを自動的に送受信することができます．このときシーケンス・トグル・ビットはM66596が自動的にトグルするので，マイコンが関与する必要はなく，連続して転送されます．

● ステータス・ステージ

ステータス・ステージは，データの転送方向がデータ・ステージと反対になりますが，制御方法はデータ・ステージと同じです．データとしてはZero-Lengthパケットを転送します．

ここまでが通常のコントロール転送の制御方法です．コントロール転送は3ステージあるため，一見すると複雑に見えますが，ステージの一つ一つの処理は難しくありません．

● ストールを受信した場合

次に，転送中にストールや通信エラーが発生した場合のM66596の動作を示します．

データ・ステージまたはステータス・ステージで，ストールを受信した場合，M66596はトランザクションの発行を停止して割り込みを発生させます．この割り込みによって，マイコンはコントロール転送の中断を判断できます．

● 通信エラーが発生した場合

M66596は通信エラーを検出すると，次の割り込みを発生します．この割り込みによってマイコンはリトライをかけて通信の継続を試みることができます．

- ペリフェラル機器側の無応答，破損パケットの受信
- INトランザクション時に最大パケット・サイズを超えたパケットを受信した場合

4 M66596によるデータ転送

M66596のデータ転送はコントロール転送に比べて，さらに簡単に制御することができます．

使用する通信PIPEごとに表2に示す設定を行い，表3の条件を満たせばトランザクションが発生します．使用するエンドポイントを設定し，転送許可を行うと，OUT方向の場合はバッファに送信データを書き込むことによりトランザクションが発行されます．また，IN方向の場合はバッファに空きがあればトランザクションが発生します．

なお，制御マイコンへは，転送の終了（バッファ・レディまたはショート・パケットの受信）やエラーの発生時に割り込みを発生して通知します．

表2 おもな設定項目

設定項目	設定内容
転送タイプ	バルク，アイソクロナス，インタラプト（コントロール転送は専用のパイプのため設定不要）
方向	IN または OUT
エンドポイント番号	EP0〜15
USBアドレス	0〜3を設定可能
マックス・パケット・サイズ	接続したペリフェラル機器のディスクリプタ情報から取得
シーケンス・トグル・ビット	最初の転送時にDATA0を指定
転送インターバル	アイソクロナス，インタラプト転送時のフレーム・インターバル
バッファ	各通信PIPEに割り当てるバッファ・サイズの指定．ダブル・バッファ，連続転送指定可能

表3 トランザクション発行条件

転送タイプ	方向	トランザクション発行条件
バルク転送	IN	バッファに空きがある場合
	OUT	バッファに送信データがある場合
インタラプト転送	IN	バッファに空きがあり，かつ転送インターバルの場合
	OUT	バッファに送信データがある場合，かつ転送インターバルの場合
アイソクロナス転送	IN	転送インターバルの場合．バッファの状態に関わらず発行される
	OUT	転送インターバルの場合．バッファの状態に関わらず発行される

● 高速なデータ転送

M66596は，ハイ・スピード対応のUSBコントローラです．高速にデータを転送するために，次の特徴を備えています．

▶フレキシブルなバッファ設定

パイプに対して，使用するバッファ・サイズを設定可能です．接続されたペリフェラル機器のディスクリプタを読み取り，高速なエンドポイントに大容量を割り当てることが可能です．

▶DMAインターフェース

M66596は，DMAインターフェースとして，DREQ，DACKの信号を2チャネル装備し，2本のパイプに割り当てることが可能です．

● DMAを用いたデータ転送

DMAを用いたデータ転送（バルクIN）の例を図9に示します．

ソフトウェアで転送を開始した後は，マイコンは転送の最後の割り込みを待つだけとなります．最大パケット・サイズやバッファ・サイズごとに割り込みを処理する必要はありません．

この例では，転送の最後がショート・パケットで終了しています．ショート・パケットを受信すると，M66596はトランスファの終了と判断して，バッファにまだ空きがあってもDMA転送を要求します．そして，M66596のバッファから受信データの読み出しが完了したときに割り込みを発生させることができます．マイコンの割り込み処理は，データの受信タイミングではなく，DMACによる転送が終了した後に動作するので，プログラミングが容易です．

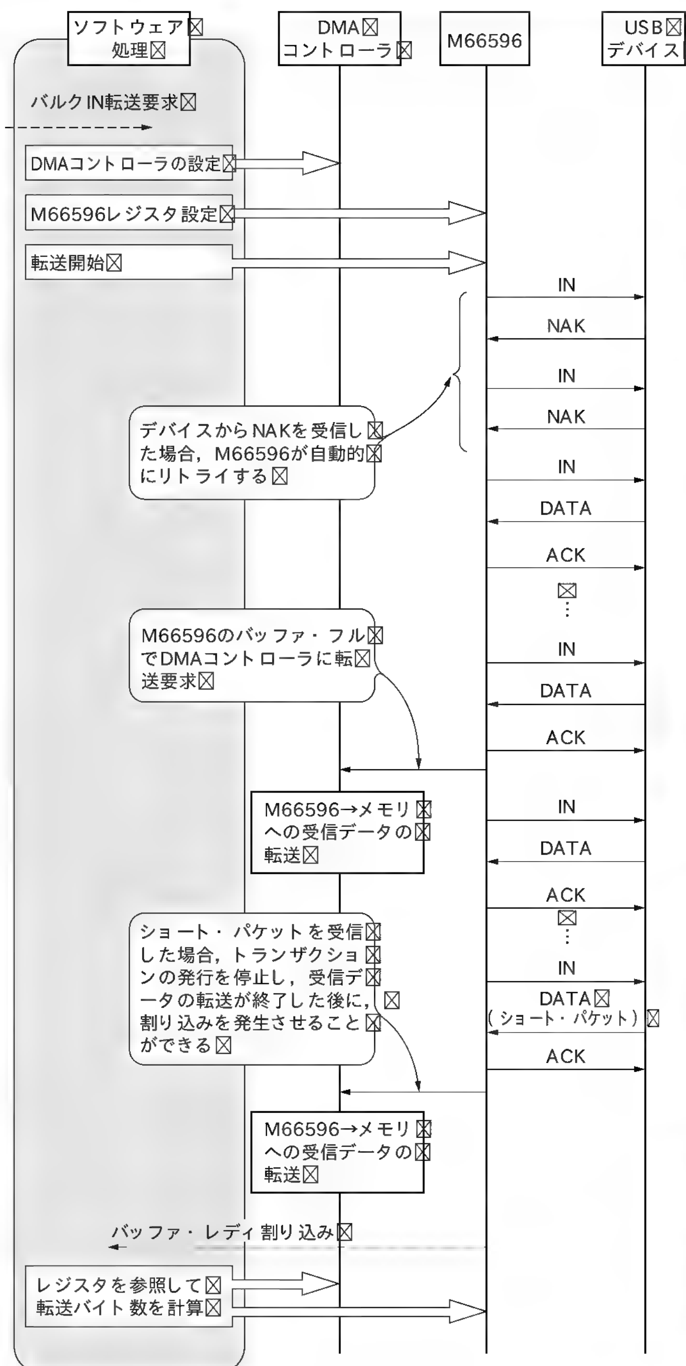


図9 DMAを用いたデータ転送（バルクIN）の例

もし，ショート・パケットが入らずに，最大パケット・サイズの転送が続く場合は，DMAコントローラに設定した転送バイト数を越えたときに，DMAコントローラから割り込みが発生することを想定しています．このタイミングで，マイコンはそれまでの転送データを処理することができます．また，あらかじめ総転送バイト数がわかっている場合には，トランザクション・カウンタ機能が使用可能です．トランザクション数を

COLUMN

ルネサステクノロジのUSBデバイス群

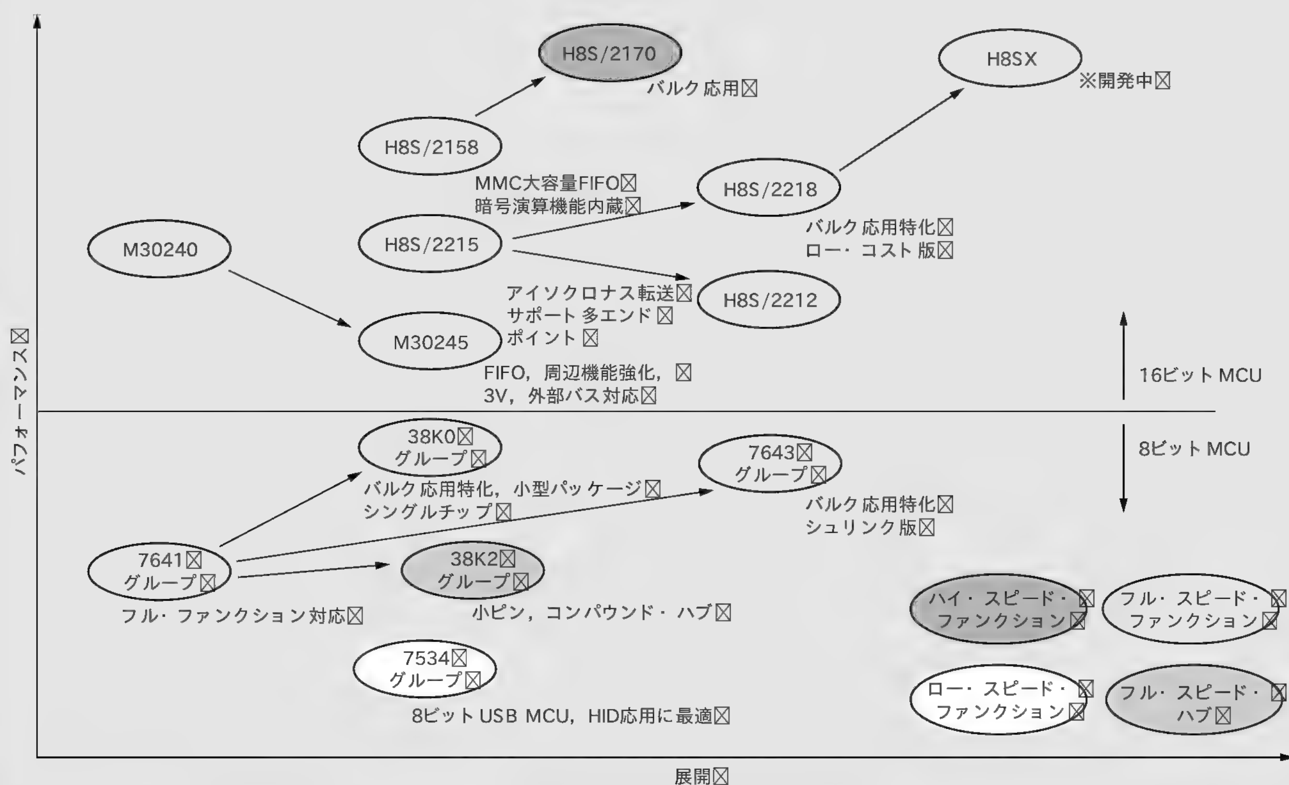
音堂 栄良

ルネサステクノロジでは，マウスやゲーム・コントローラなどの低価格HIDデバイス向けロー・スピード対応8ビット・マイコン7534をはじめ，フル・スピード対応8ビット・マイコン7641グループ/38K0/2グループ，フル・スピード対応16ビット・マイコンM16C，H8S，ハイ・スピード対応16ビット・マイコンH8S，フル・スピード対応USBターゲット＆OHCI対応ホスト内蔵SuperHまで，さまざまなUSB内蔵マイクロコントローラを準備しています．図Aおよび図Bは8/16/32ビット・マイコンの展開です．

また，フル・スピード対応M6629x ASSP，ハイ・スピード対応M6659x ASSPや，システムLSIに対応可能なUSB IPモジュールも準備しており，USBに関するさまざまなご要求に対応できます（図C）．

各USBデバイスは評価ボード，いろいろなデバイス・クラスに対応したUSBインタフェース制御プログラム，アプリケーション・ノートなどを準備しています．また，各種USB関連のセミナも開催しています．詳細は，http://www.renesas.com/jp/usb/ へアクセスしてください．

おんどう・えいりょう （株）ルネサステクノロジ



図A　USB内蔵8/16ビット・マイコンの展開

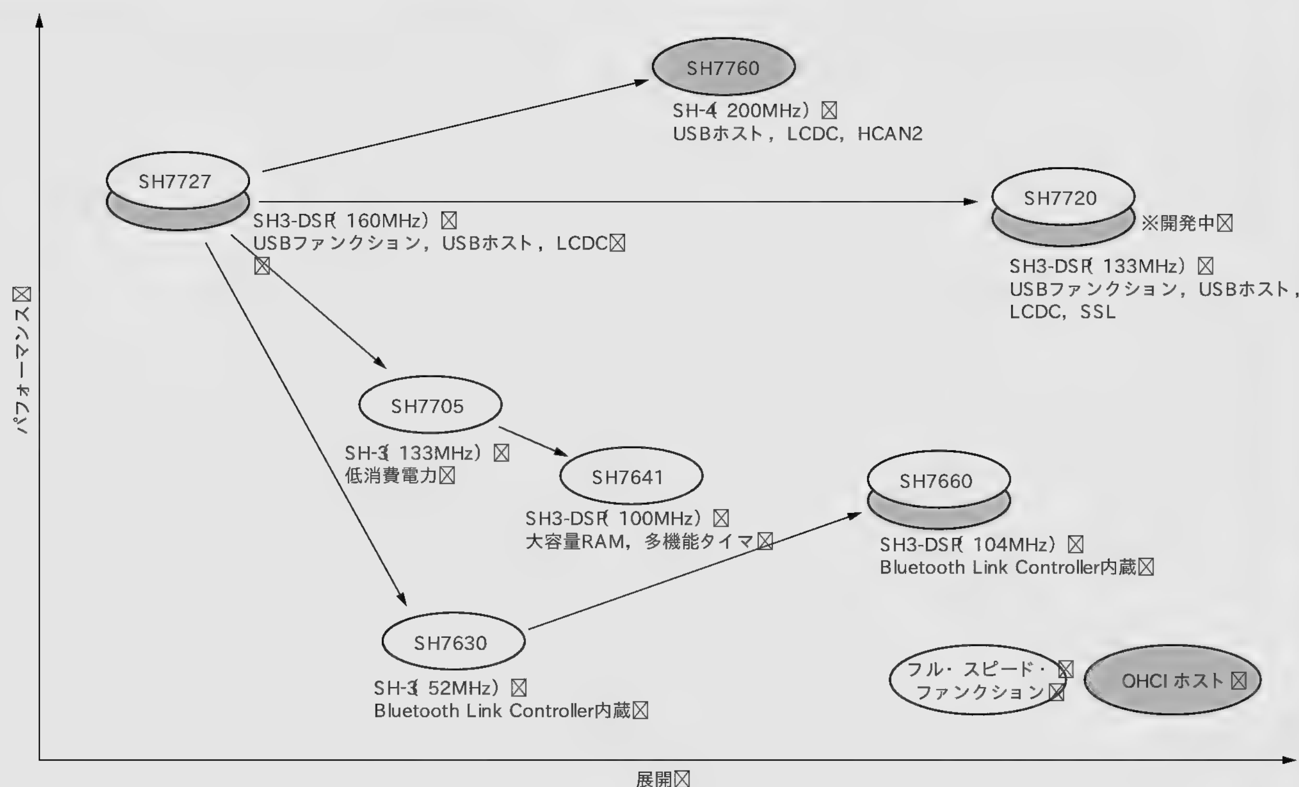
設定しておくと，その回数の読み出しが完了した時点で割り込みを発生させることができます．

このように使用するパイプへの設定をした後は，M66596のデータ通信は内蔵バッファへのリード/ライトで行います．この制御方法は，M66596をペリフェラル・モードで使用した場合も同様です．つまり，ホスト時でもペリフェラルでも同様の手順でデータ通信できるため，ホスト/ペリフェラルを切り替えるような応用の場合でも，ソフトウェアはシンプルなものになります．

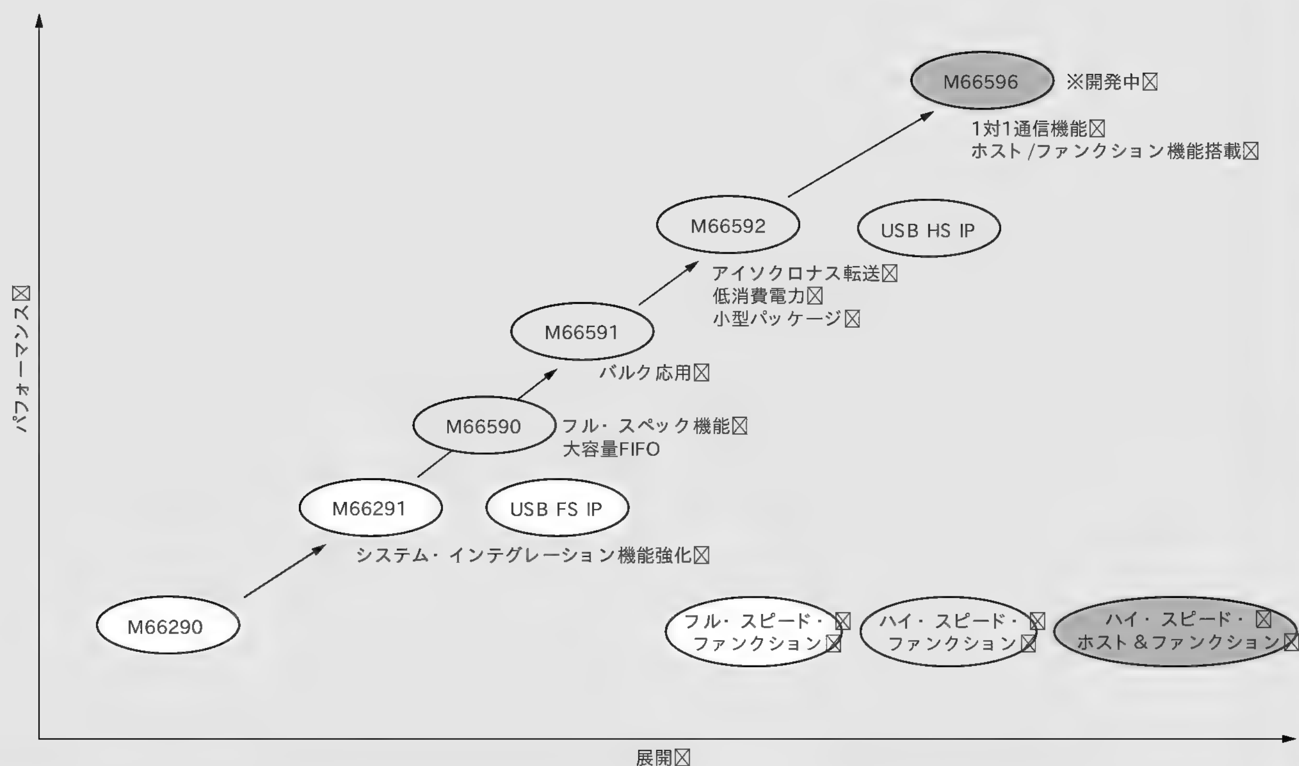
まとめ

ホスト・コントローラというと，ソフトウェアによる制御が難しいとか，システムへの組み込みがたいへんなのでは？と思う人も多いと思います．M66596はデバイス・コントローラのように，気軽に使えるUSBホスト・コントローラです．このようなチップによって組み込み機器の拡張性が増し，より魅力的な製品が市場に出てくることを期待します．

かとう・ともゆき/いえた・じゅん/ひらの・さねあき
（株）ルネサス ソリューションズ



図B　USB 内蔵 32ビット・マイコン展開



図C　USB ASSP 展開

USBターゲット機能とOHCI準拠ホスト・コントローラを内蔵した

5 On-The-Goの概要とML60842を使ったOTGシステムの開発事例

宮田 学/岡崎 真也/齊藤 孝之

第3部ではOn-The-Go対応デバイスとして，ML60842を取り上げる．デバイス内部にバッファRAMを内蔵することで，SRAMなどと同様にマイコンのローカル・バスに直結可能なOHCI（OpenHCI）仕様に準拠したホスト・コントローラを内蔵している．ここではOn-The-Go仕様の概要を解説したあと，ML60842について詳しく解説する．（編集部）

はじめに

USBは当初，PCと周辺機器との接続のための汎用バスとして開発されたため，PCのみがバス・マスタ（USBではホストと呼ぶ）として動作し，複数の周辺機器は直接もしくはハブによるスター型で接続され，バス・スレーブ（USBではペリフェラルと呼ぶ）として動作する仕様で規格が策定されました．また，USBでは接続のまちがいなどによる故障や動作トラブルを未然に防ぎ，ユーザの負担を減らすため，使用するケーブルやハブのポートに方向性があり，規定された物理トポロジ以外では接続できない仕様となっています．

しかし，USBが広く普及するにしたがい，周辺機器どうしでの相互接続に対する要求が高まってきました．その要求に応えるためにUSBの拡張という形で考案され，規格化された周辺機器のホスト動作による相互接続技術がOn-The-Go Supplement to the USB2.0 Specification（以後OTG）です．ここではこのOTG規格とそれに準拠したOTGシステムの開発事例を解説します．

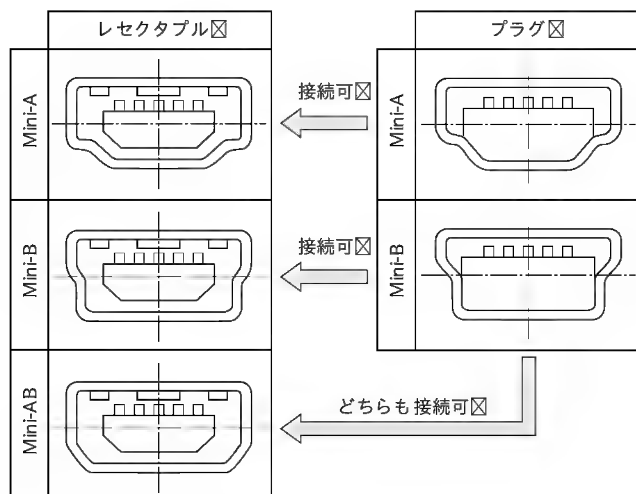


図1 コネクタとレセクタプル

1 USB On-The-Goとは何か

● USB機器の相互接続を実現

OTGでは相互接続を実現するにあたり，デバイスにペリフェラル機能だけでなくホスト機能が必要になります．このホスト/ペリフェラル両方の役割をもったOTGデバイスを，デュアル・ロール・デバイスと呼びます．OTGはUSBの拡張という位置づけのため，バスはシングル・マスタを前提としています．そして，この前提に対する互換性を維持し，コスト面，構造上の実現容易さなどを考慮した結果，OTG機器どうしの接続はピアtoピア接続を主体として規格化されています．そのため，PCをホストとしてスター型に構成されたバス上にデュアル・ロール・デバイスを接続してもホストとして動作することはできませんが，デュアル・ロール・デバイスをホストとしたスター型のバス構成は許されており，これを保証するためのコネクタとケーブルの組み合わせも規定されています．

また，デュアル・ロール・デバイスの要件としてSession Request Protocol（以下SRP）と，Host Negotiation Protocol（以下HNP）という二つのプロトコルのサポートがあります．SRPはバスを使用するときのみアクティブにするためのプロトコルであり，HNPはピアtoピア接続時にホスト，ペリフェラルの役割を入れ替えることで，ケーブル接続方向に縛られずホスト動作を可能にするプロトコルです．以降ではこれらOTG規格の要所を個別に説明します．

● コネクタ，ケーブル

OTGでは，コネクタの小型化，兼用による省スペースと簡単な接続をめざして，Mini-Aプラグ&レセクタプル（プラグはケーブル側，レセクタプルは基板側コネクタ），Mini-ABレセクタプルが規定されました（図1）．

表1 ケーブルとアダプタの種類

		標準Aプラグ	Mini-Aプラグ
ケーブル	標準Bプラグ	○	○
	Mini-Bプラグ	○	○
アダプタ	標準Aレセクタプル	−	○
	Mini-Aレセクタプル	○	−
	Mini-ABレセクタプル	−	−

Mini-Aプラグは小型のAプラグであり，大きさはUSB2.0のECNとして追加されたMini-Bプラグと同等で，同様にID端子を持っています．ただし，ID端子はMini-Bと異なりGNDに接続されています．レセクタプル側でそのID端子の電位により，挿入されているプラグの種類が判別可能になります．

Mini-Aレセクタプルは，Mini-Aプラグのみを挿入可能なレセクタプルです．

Mini-ABレセクタプルは，Mini-AプラグとMini-Bプラグの両方が挿入可能なA，B兼用のレセクタプルになっています．デュアル・ロール・デバイスにはこのMini-ABレセクタプルを搭載することが要件となっています．そして，デュアル・ロール・デバイスはMini-ABレセクタプルに挿入されたプラグの種類（ID端子により判別する）によってMini-AならAデバイス，Mini-BならBデバイスとして動作します．

加えて，Aデバイス，Bデバイスは，デフォルト・ロール（Aならホスト，Bならペリフェラル）が決まっています．デフォルト・ロールと呼ばれる理由は，HNPによりケーブル接続を変更せずにホストとペリフェラルの役割（ロール）を入れ替えることが可能であるため，初期状態での動作を区別するためです．また，HNPにより役割を入れ替えても，電源供給はつねにAデバイスが担います．

この三つのコネクタと従来のコネクタとの組み合わせで，さまざまな種類のケーブルが考えられますが，接続のまちがいなどを極力回避するために，表1の必要な組み合わせのケーブル，アダプタのみに種類は限定されています．これにより，コネクタの大きさに違いはありますが，AとBコネクタの組み合わせのケーブルしかなく，方向性が必ず存在することになります．したがって，A，B兼用のMini-ABレセクタプルを持つデュアル・ロール・デバイスどうしでの接続でも必ずAデバイス（＝デフォルト・ホスト）とBデバイス（＝デフォルト・ペリフェラル）での接続となります．また，デュアル・ロール・デバイスをPCに接続する場合でも，PC側はAプラグしか挿入できないので，デュアル・ロール・デバイスはMini-Bプラグでの接続となり，ペリフェラルとしてのみ動作することになります．

● SRP

OTGでは省電力化のため，通信しないときはバスを停止することが要求されます．この際，V_{BUS}による電源供給も停止します．そのため，通信を開始したいデバイスがAデバイスの場合は，ポート・パワー（V_{BUS}）をONにすればバスをアクティブにできますが，Bデバイスの場合は，AデバイスにV_{BUS}供給を開始してもらわなければなりません．それを実現するプロトコルがこのSRPです．

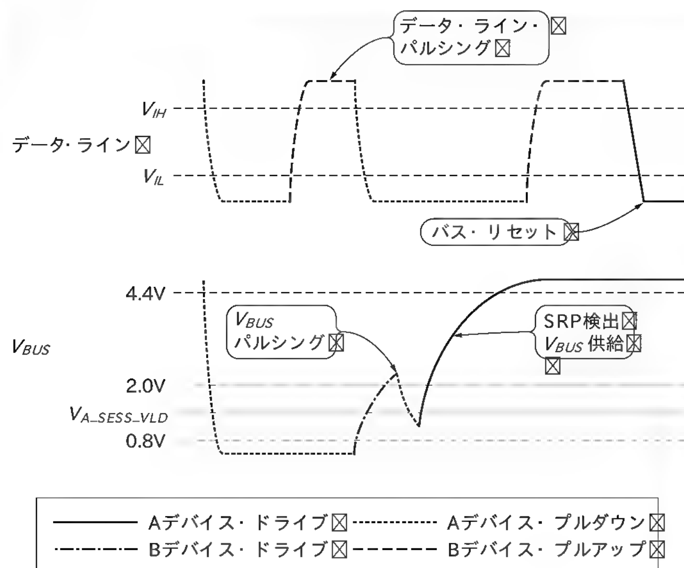


図2 SRPシーケンス

SRPは図2のように，Bデバイスよりデータ・ラインとV_{BUS}に対してパルス信号を送信し，それを検出したAデバイスがV_{BUS}供給を開始します．SRPの要件としてBデバイスは，データ・ライン・パルシング，V_{BUS}パルシングの両方をサポートしなければならず，Aデバイスはデータ・ライン・パルシング，V_{BUS}パルシングの少なくともどちらか一方を検出できなければなりません．

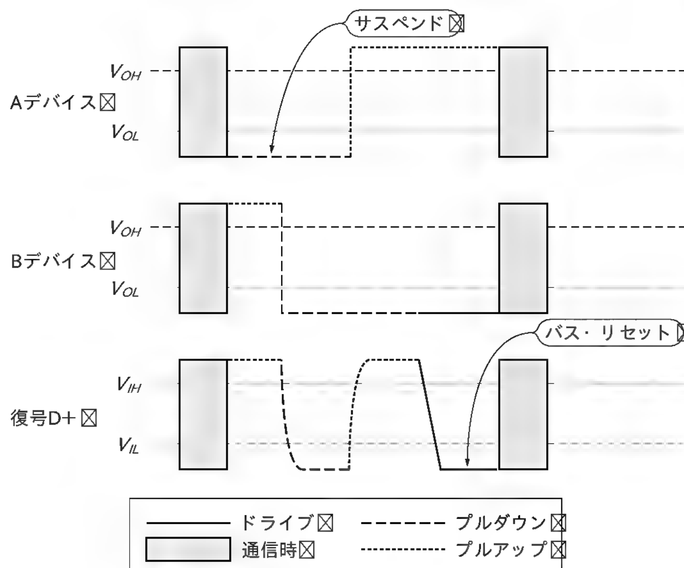


図3 HNPシーケンス

COLUMN

On-The-Goとホスト仕様

OTGを実現するには，ホスト機能とペリフェラル機能，そして，それを切り替え制御するOTG機能の三つの機能が必要です．しかし，ホスト機能とペリフェラル機能については，それぞれのレジスタ仕様などの細かな規定はOTGの中にはありません．つまり，実際にデバイスに実装するホスト・コントローラとしては，UHCIやOHCIに準拠したものでも，独自のホスト・コントローラでもかまわないのです．

とはいえ，ホスト機能もペリフェラル機能も，OTG機能部と連携して動作できなければならないため，たとえばASICを起こすにしても，OHCIのコアを買ってきて貼り付ければそれで完成…というわけにはいきません．

● HNP

前述したようにデュアル・ロール・デバイスには，A，B兼用のレセクタプルが搭載されているので，どちらの方向でもケーブルは接続できます．しかし，ケーブルの接続方向のみでホスト，ペリフェラルの役割が決まってしまう場合，ユーザはケーブルの接続に注意を払わなくてはなりません．そのため，ピアtoピア接続時に限りケーブルの接続方向を変えずにホスト，ペリフェラルの役割を入れ替えることができるプロトコルとしてHNPが規定されています．

図3(p.109)に，AデバイスからBデバイスへのホスト切り替えを例にHNPのシーケンスを示します．このプロトコルは，図3のようにホストがUSBバスをサスペンドすることをきっかけに開始され，ペリフェラル動作を行っていたデバイスがデータ・ラインのプルアップを止めることでディスコネクトすると，

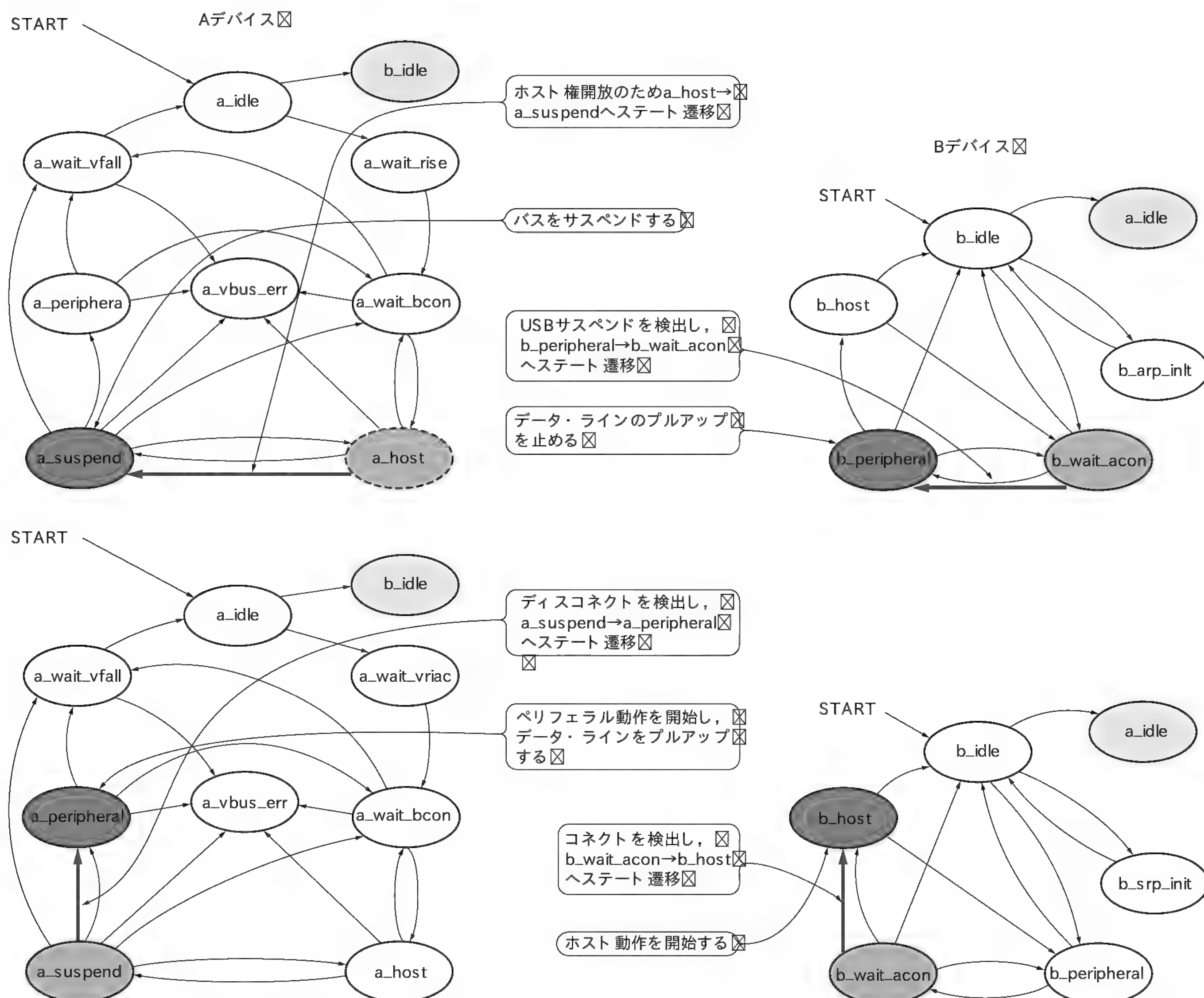


図4 AデバイスからBデバイスへホスト変更の流れ

ホスト動作を行っていたデバイスが今度はデータ・ラインをプルアップし，それぞれの役割の入れ替えが完了します．

OTGではA，Bデバイスそれぞれのステート・マシンにより，ホスト，ペリフェラルの切り替え動作を定義しています．各ステート・マシンでは，ステートを遷移する際の遷移条件と各ステートでのUSBコントローラの動作が決められています．

図4(p.110)に，HNPによるAデバイスからBデバイスへのホスト切り替えを例に，両デバイスのステート遷移を示します．

● その他

その他の規定としては，次のものがあります．

- 限定されたホスト機能
- ターゲット・ペリフェラル・リスト
- メッセージ出力(No Silent Failures)

OTGのホスト機能は，PCホストに求められるすべての機能をサポートする必要はありません．ペリフェラルに対する供給電流は8mA以上であれば良く，ハブをサポートしない，もしくはハブの階層やポート数に制限をつけることも可能です．また，ホスト動作時にサポートするデバイスの種類も，申告したターゲット・ペリフェラル・リストに列挙されているデバイスのみでOTGデバイスとしての要件を満たすことになります．

No Silent Failuresと呼ばれる規定は，デバイスがユーザに何も伝えずに停止してはならないといった決まりであり，デバイスと接続を確立できないときには，ユーザにその旨を伝えなければなりません．たとえば，サポートしていないデバイスが接続された，ハブによるカスケード接続をサポートしていない，SRPに応答がなかったなどのメッセージ表示を行う必要があるのですが，具体的なメッセージの文言や表示方法までは規定されていません．

2 On-The-Goコントローラ ML60842の概要

次に，OTGシステム開発に使用したOTGコントローラML60842(沖電気工業)について解説します．

ML60842は，OHCI 1.0a準拠のUSBホスト・コントローラML60852(同社)と同じフル・スピードUSBデバイス・コントローラ，およびUSBトランシーバを内蔵したコントローラです．その内部構成を図5に，特徴を表2に示します．

● USB On-The-Go機能

ML60842のOTGコントローラ部は，ID端子状態検出機能，V_{BUS}電圧検出機能，V_{BUS}パルス出力機能をもっています．V_{BUS}の電源供給機能については，チャージ・ポンプを搭載していないため，外部回路として5V電源とV_{BUS}用電源スイッチICなどが必要になります．

▶ID端子状態検出

OTGで追加されたIDピンの変化を検出することができ，IDピンが変化するとCPUへの割り込みを発生します．

▶V_{BUS}電圧検出

OTGを実現するために必要な電圧レベルを検出するためのコンパレータを内蔵し，V_{BUS}電圧がコンパレータに設定した電圧をまたいだときにCPUへ割り込みを発生します．

▶V_{BUS}パルス出力

V_{BUS}端子をプルアップする/プルダウンする機能があり，V_{BUS}をプルアップして一定時間後に，プルダウンするとパルスが生成されます．

● ペリフェラル機能

ペリフェラル・コントローラは，フル・スピードに対応するML60852(沖電気工業)相当のコントローラが内蔵されています．

ML60842の使用可能なエンドポイント数は，ML60852と同

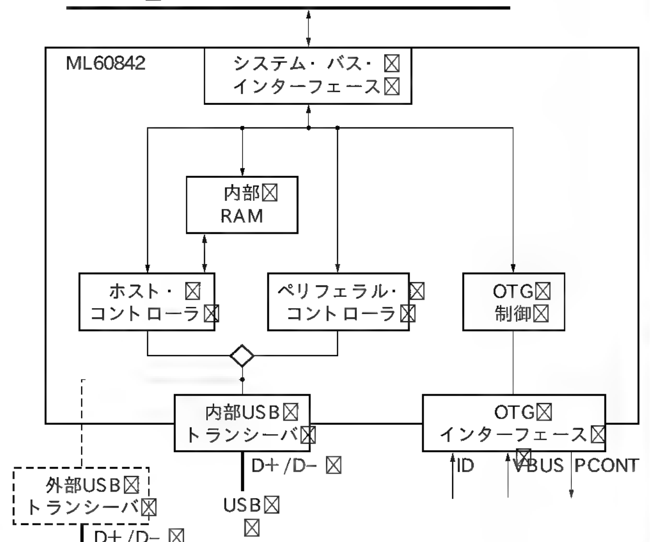


図5 ML60842ブロック図

表2 On-The-Goコントローラ ML60842の特徴

On-The-Go機能
● ID端子状態検出 ● V_{BUS}電圧検出 ● V_{BUS}パルス出力
ホスト機能
● フル・スピード(12Mbps)，ロー・スピード(1.5Mbps)に対応 ● OHCIに準拠したコアを内蔵 ● 4Kバイトの専用RAMを内蔵 ● スレーブDMA 1チャネル
ペリフェラル機能
● フル・スピード(12Mbps)対応 ● エンドポイント数=5個あるいは6個 ● EP1，EP2，EP4，EP5のFIFOは2面構成 ● スレーブDMA 2チャネル
その他
● 16/8ビット・バス幅切り替え可能 ● アドレス/データ・マルチプレクスに対応 ● リトル/ビッグ・エンディアン切り替え可能 ● パワー・ダウン機能サポート ● 3.3V単一電源

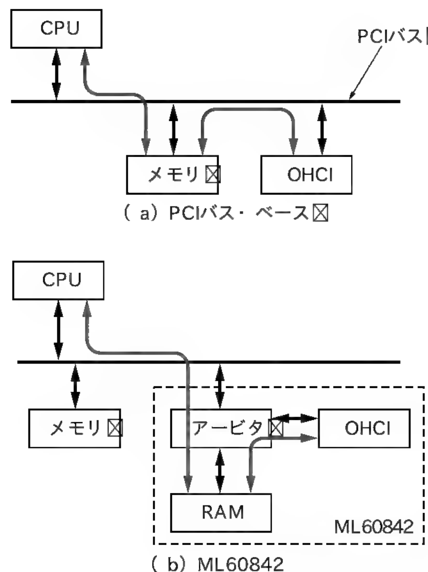


図6 OHCIホスト・コントローラ

じで，次のようになっています．

- コントロール転送専用エンドポイント×1個
- バルク/インタラプト転送用エンドポイント×3個
- アイソクロナス/バルク/インタラプト転送用
- エンドポイント×1または2個

各エンドポイントのFIFOは2面構成となっています．そのため，エンドポイントに割り当てられた一方のFIFOがデータ転送中であっても，もう一方のFIFOに次の転送用データを準備することが可能であり，USBのバンド幅を有効に利用できます．

● USBホスト機能

ML60842は，PC向けのOHCIコントローラと同等のコアと4KバイトのRAMを内蔵しています．

通常のOHCIではシステム・バスがPCIですが，ML60842では組み込み用途を想定し，SRAMアクセス・ライクなシステム・バスを採用しています．

通常のOHCIでは，図6(a)に示すようにPCIバスに接続することを前提としています．CPUは転送に必要なディスクリプタ(後述)と転送するデータをメモリ上に置きます．OHCIはPCIバス・マスタとして動作し，メモリ上のディスクリプタとデータを読み出してUSBへ転送します．

しかし，多くの組み込み向けのCPUがPCIバスを持っておらず，また，外部のバス・マスタが動作するために，システム・バスを開放する機能をもっていません．そのためML60842では，図6(b)に示すように内部に専用のRAMを設け，CPUが転送に必要なディスクリプタと転送するデータをその内部RAM上に置けば，OHCIがバス・マスタとして動作するのと同様に，内部RAM上のディスクリプタとデータを読み出してUSBへ転送します．

これにより，ML60842は組み込み向けのCPUと容易に接続することができ，高機能なUSBホスト・システムを構築することが可能です．

● 接続デバイス数とエンドポイント数

USBコントローラで気になるものの一つとして，接続可能なデバイス数や使用可能なエンドポイント数があります．OHCIの論理的に接続可能なデバイス数や使用可能なエンドポイント数は，USB仕様で定められている数になります．しかし，ML60842はOHCIコアを採用していますが，転送に必要なディスクリプタと転送するデータを内蔵RAM上に置かなくてはなりません．ML60842では内部RAMが4Kバイトと制限されるため，現実的には使用可能なエンドポイント数は十数個となり，接続可能なデバイス数は数台となります．これは，ML60842のUSBホストがOTGのホストと考えれば十分な数です．

● PCIバス上のOHCIとの違い

ML60842にはPCIコンフィグレーション・レジスタはありません．デコードされたチップ・セレクト信号(CS)を接続すると，マッピングされた空間からOHCIの制御レジスタと内部RAMへアクセスすることができます．また，バス・マスタとして動作せず，スレーブDMAインターフェースを持ちます．

ML60842は，OHCI制御レジスタとは別に内部RAMのアドレスを設定するレジスタがあります．このレジスタには，システム上にマッピングされた内部RAMの物理アドレスを指定します．ML60842内部のOHCIがメモリへアクセスするときに，このレジスタに設定されたアドレスとアクセスするアドレスを比較します．アクセスするアドレスが内部RAM上のアドレスであれば，そのまま内部RAMをアクセスします．アクセスするアドレスが内部RAM上のアドレスではないとき，ML60842はCPUへ割り込みを発生します．割り込みが発生したとき，OHCIがアクセスするアドレスや転送方向がレジスタにセットされるので，CPUはそれにしたがってマスタDMAを起動します．このように，システム・バス接続されたメモリ上にあるデータを直接転送することもできます．

3 Open Host Controller Interface (OHCI)の概要

ここでML60842がホスト・コアとして内蔵しているOHCIについて簡単に説明します．

OHCIにはコミュニケーション・チャネルとして，制御レジスタとホスト・コントローラ・コミュニケーション・エリア(HCCA)があります．制御レジスタはコントロール，ステータスおよびリスト・ポインタなどのレジスタを含みます．

制御レジスタの中にHCCAと名前を付けられた共有メモリのロケーションへのポインタがあります．HCCAはインタラプト転送のエンドポイント・ディスクリプタ(ED)のリストへのポインタと，処理が終了した転送ディスクリプタ(TD)のキューへのポインタ，およびフレーム処理に関するステータス情報を含みます(図7)．

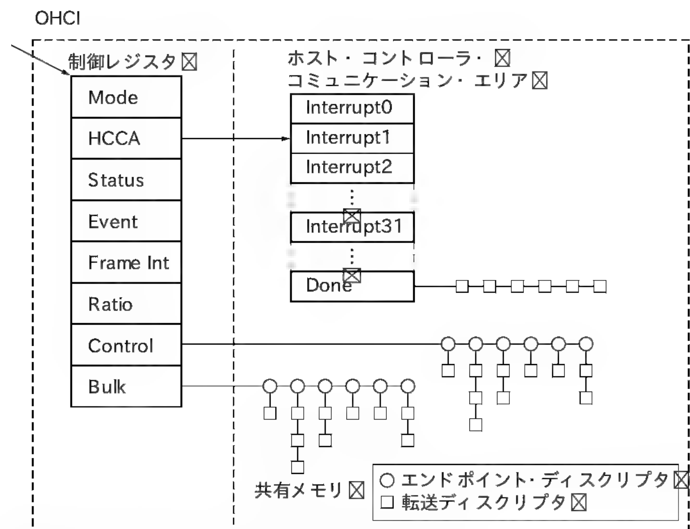


図7　コミュニケーション・チャネル

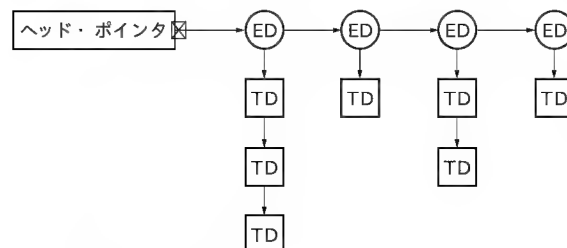


図8　標準リスト構造

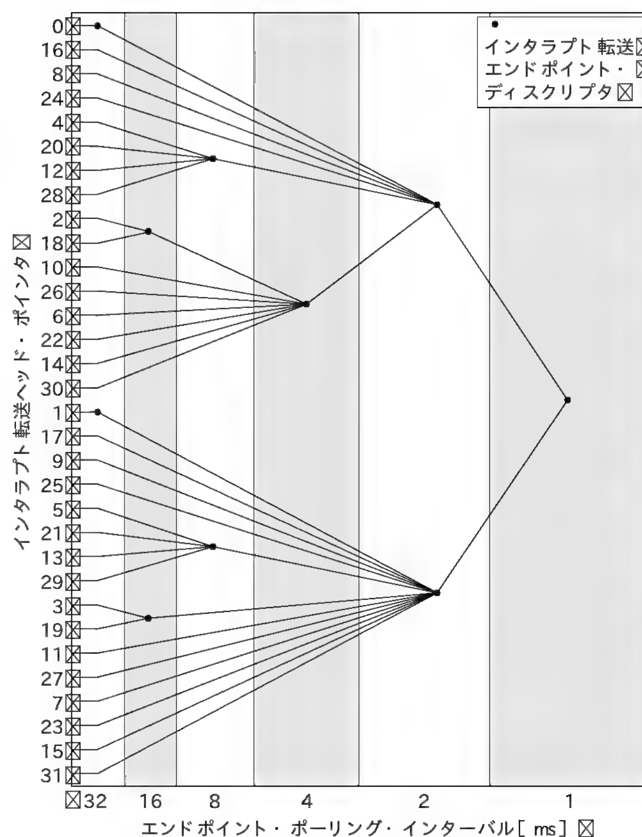


図9　インタラプト転送エンドポイント・スケジュール例

● データ構造

コミュニケーション・インターフェースはエンドポイント・ディスクリプタ(ED)と転送ディスクリプタ(TD)で構成します．

エンドポイント・ディスクリプタは，そのエンドポイントの最大パケット・サイズ，エンドポイント・アドレス，データ転送速度，およびデータ転送方向の情報をもちます．複数のエンドポイント・ディスクリプタはリスト構造によりリンクされます．

転送ディスクリプタのキューはエンドポイント・ディスクリプタからリンクされます．転送ディスクリプタは，データ・トグル情報，データ・バッファ・ロケーション，および完了ステータス・コードの情報をもちます．一つの転送ディスクリプタは一つ以上のデータ・パケットの情報を含みます．転送ディスクリプタのキューはリスト構造でリンクされ，その最初にリンクされたキューが最初に処理されます．

USBのデータ転送タイプごとに処理されるエンドポイント・ディスクリプタのリストがあります．図8は標準的なリスト構造関係の表現です．

コントロール転送とバルク転送のエンドポイント・ディスクリプタのリストへの先頭ポインタは制御レジスタの中の各ヘッダ・レジスタに保持させます．

インタラプト転送のエンドポイント・ディスクリプタのリストへのヘッド・ポインタは，HCCA内に保持させます．アイソクロナス転送のエンドポイント・ディスクリプタのリストへのヘッド・ポインタはありません．アイソクロナス転送のエンドポイント・ディスクリプタは，インタラプト転送のエンドポイント・ディスクリプタのリストの最後にリンクします．

HCCAには32個のインタラプト転送のエンドポイント・ディスクリプタへのヘッド・ポインタがあり，フレーム・カウンタにより処理されるヘッド・ポインタが決定されます．

インタラプト転送のエンドポイント・ディスクリプタは，ヘッド・ポインタをリーフ・ノードとしたツリーを構成します．

図9はインタラプト転送エンドポイントのスケジュール例です．このスケジュールは，1msのポーリング間隔で二つのエンドポイント・ディスクリプタを示します．2msのポーリング間隔での二つのエンドポイント・ディスクリプタ，4msのポーリング間隔での一つのエンドポイント，8msポーリング間隔での二つのエンドポイント・ディスクリプタ，16msのポーリング間隔で二つのエンドポイント・ディスクリプタ，および32msのポーリング間隔で二つのエンドポイント・ディスクリプタがあることを示します．

● エンドポイント・ディスクリプタ(ED)

エンドポイント・ディスクリプタ(ED)は，16バイト(メモリ上の16バイト・バウンダリに配置する必要がある)構造です．エ

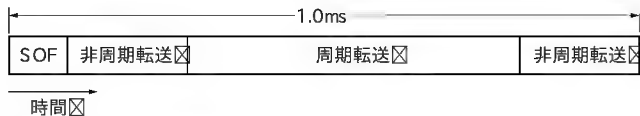


図10　フレーム帯域幅割り付け

ンドポイント・ディスクリプタは，エンドポイント・アドレス，最大パケット・サイズおよびトグルビットなどエンドポイントに必要な情報を保持します．ホスト・コントローラは，EDのリストを検索します．EDとリンクされたTDがある場合，ホスト・コントローラはTDによって指示された転送を実行します．

● 転送ディスクリプタ(TD)

転送ディスクリプタ(TD)は，エンドポイントに転送するデータのバッファを定義します．TDには汎用TDとアイソクロナスTDの二つのタイプがあります．汎用TDはインタラプト転送，コントロール転送およびバルク転送のエンドポイントに使用します．汎用TDは16バイト，アイソクロナスTDは32バイトの構造です．アイソクロナスTDはアイソクロナス転送に使用します．汎用TDおよびアイソクロナスTDはともに，0～8192バイトのバッファを指定することができます．

USBのデータ転送は，EDのリストにリンクされた転送ディスクリプタのキューにより実現されます．

● リスト管理

データ転送を制御するためには，EDをリストに加えたり，リストから取り外さなければなりません．リストに加える場合はリストの最後にEDを加えます．この場合，リスト処理しているホスト・コントローラとの競合を考慮せずに追加ができます．EDをリストから取り外す場合，ホスト・コントローラ・ドライバはホスト・コントローラが削除するEDにアクセスしていないことを保証するために，そのEDの処理を停止する必要があります．

また，データ転送を制御するためにTDをキューに加えたり，取り外さなければなりません．キューに入れるにはエンドポイントのキューの最後にTDを加えます．この場合も，キューを処理しているホスト・コントローラとの競合を考慮せずに追加ができます．正常なオペレーションでは，ホスト・コントローラがデータを転送した後，TDをキューから外し，処理完了キューとTDをリンクします．TDは正常完了またはエラー条件により取り除かれます．TDが取り除かれたとき，コンディション・コードがホスト・コントローラによりセットされ，ソフトウェアでTDが取り除かれた理由を得られるようになっています．TDが上位ソフトウェアからのリクエストにより取り消されるか，またはエラーにより停止している場合には，ソフトウェアでTDをキューから外します．

このように，OHCIではメモリ上のEDとTDをリンクして，USBのデータ転送を制御します．

● 帯域幅割り付け

OHCIの帯域幅割り付けの方法を図10に示します．各フレームはホスト・コントローラがUSBへ同期パケット(SOF)を送ることで始まります．帯域幅の一部分は非周期転送のために確保されます．一定量のコントロール転送とバルク転送が各フレーム内に生じることを保証します．フレーム・インターバル・カウンタはホスト・コントローラが周期転送を開始する時間を示し，セットされた値に達するまで，ホスト・コントローラは非周期転送を実行します．フレーム・リメイン・カウンタがフレーム・インターバル・カウンタ値に達すると周期転送を実行します．そのフレーム内の周期転送が終了した後，フレームの残り時間は非周期転送を実行します．

ソフトウェアで周期転送の各エンドポイントに利用可能な帯域幅を分配します．十分な帯域幅が利用できない場合，新しく接続した周期転送のエンドポイントへはバスへのアクセス権を与えないようにします．

● ルート・ハブ制御

ルート・ハブ機能はホスト・コントローラに統合されています．ホスト・コントローラの制御レジスタにルート・ハブを制御するために必要なレジスタが含まれます．ソフトウェアにてハブ・クラスのプロトコルに準じてルート・ハブの適切な応答を返す必要があります．

4 On-The-Goデバイス機器の開発事例

ML60842を使用したOTGデバイス機器として，USBマスストレージ・クラス・ドライバを実装したOTGマスストレージ・デモ・システムを開発しました．ここではこのOTGマスストレージ・デモ・システムのハードウェアおよびソフトウェアの機能，構成，デュアル・ロール・デバイスの開発手法について説明します．

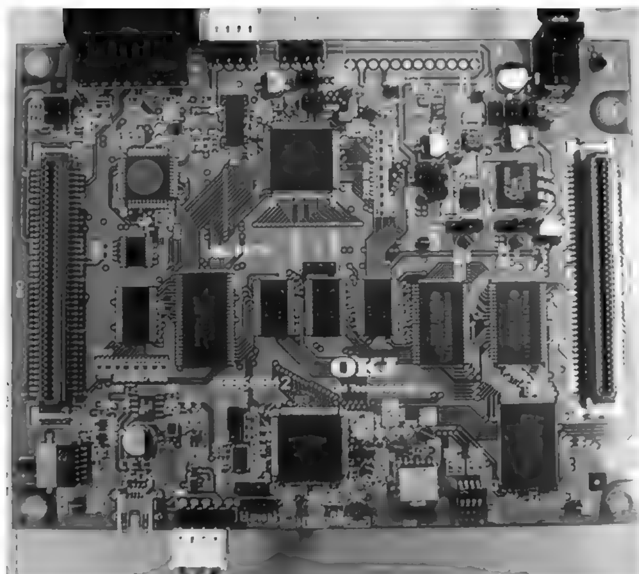
● ハードウェア構成

写真1に今回のデモ・システムの外観を示します．デモ・システムを動作させるハードウェアには，JOB60842ボード(沖電気工業)を使用しています．JOB60842ボードは，ARM7TDMIをCPUコアとして搭載しているμPLAT-7BをCPUプラットホームとして採用した沖電気工業製32ビット・マイコンML674000とOTGコントローラML60842を搭載した，USB2.0 OTGシステムを開発するためのスタータ・キットです．USBマスストレージ・デモ・システムは，このJOB60842ボードを使用して開発しました．

図11にJOB60842ボードのブロック図を，表3にJOB60842ボードの仕様を示します．

● OTGドライバ概要

OTGドライバはOTGに準拠し，SRP，HNPをサポートするDRD(デュアル・ロール・デバイス)ドライバです．OTGドライバはML60842の共通部を制御し，USBのV_{BUS}(ON/OFF)制


写真1　デモ・システムの外観

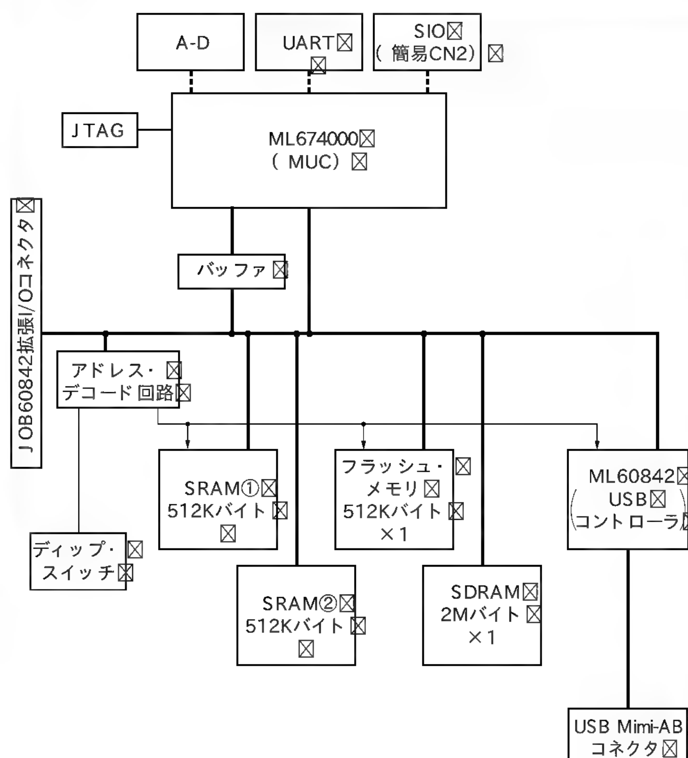


図11　JOB60842ボードのブロック図

御，データ・ライン・プルアップ（接続/切断）制御，ホスト/ペリフェラルのロール切り替えを制御するドライバです．OTGドライバを組み込むことによって，デバイスは従来固定されていたホスト/ペリフェラルと電源供給/消費（Aデバイス/Bデバイス）の役割を変更することが可能となります．OTGドライバはNORTi Ver.4（ミスポ）上で動作します．

● OTGドライバ・タスク構成

OTGドライバは一つのタスクから構成され，OTGドライバが提供するAPIによってイベントの発生を各モジュールからOTGドライバへ通知します．通知を受信したOTGドライバは，通知の種別に従って処理をします．図12にOTGドライバ・タスクと，ほかのモジュールのタスクとの関係を示します．

● OTGドライバ動作概要

OTGドライバには，OTG仕様に準拠したステート・マシンが実装されています．OTGドライバはOTGイベントをAPIから受け取ると，それに対応した処理を行うステート・マシンとして動作します．OTGドライバへイベントの発生を通知するタスクおよびハンドラは，次のとおりです．

●アプリケーション・タスク

アプリケーションからセットするOTGイベントは，接続要求に対するイベントです．OTGドライバは，アプリケーションからのUSBの接続要求がある場合のみ，USBの動作を開始します．

●ホスト・ドライバ

ホスト・ドライバからセットするOTGイベントは，ペリフェラルの接続，切断イベントと，HNPに関するイベントです．

●ペリフェラル・ドライバ

ペリフェラル・ドライバからセットするOTGイベントは，HNPに関するイベント，データ・ラインの状態変化，デバイスのコンフィグレーション状態に関するイベントです．

●割り込みハンドラ

割り込みハンドラから通知するイベントは，ML60842共通部が検出するOTGイベント（V_{BUS}の状態変化，Aデバイス，Bデバイスの切り替え，SRPの検出）です．

●タイマ・ハンドラ

タイマ・ハンドラから通知するイベントはOTGステート・マシン内でセットされるタイマによって発生する処理タイムアウト・イベントです．

表3　JOB60842ボードの仕様

ML674000（MCU）	動作周波数	32.0MHz
	内部RAM	8Kバイト
	SIO	簡易コネクタ実装
	UART	未実装（簡易コネクタ）
	JTAG	実装（20ピン）
外部メモリ	SRAM	プログラム・ロード用512Kバイト
		プログラム・ワーク用512Kバイト
	フラッシュ・メモリ	ブート・プログラム・セーブ用512Kバイト
	SDRAM	2Mバイト
ML60842（USBコントローラ）	動作周波数	48.0MHz（水晶振動子）
	バス・クロック	32.0MHz
	内部RAM	4Kバイト（USBホスト動作時）
	USBホストOTG	USB Mini-ABレセクタプル実装

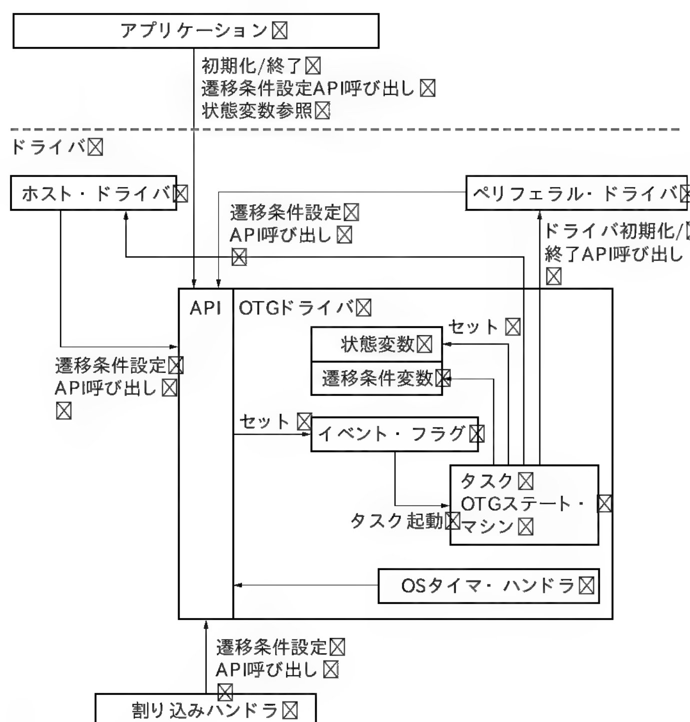


図12　OTGドライバの位置づけ

● ホスト・ドライバ/ペリフェラル・ドライバの動作

ホスト・ドライバおよびペリフェラル・ドライバが，OTGデバイスとして動作するときの動作概要を説明します．

▶ ホスト・ドライバ

ホスト・ドライバは，ホスト動作時にOTGドライバによって起動されます．ホスト・ドライバは，ホスト動作時は次のイベントが発生した際にOTGドライバへイベントの通知をします．

●ルート・デバイス検出

ホスト・ドライバはルート・デバイスの接続を検出したとき，OTGドライバへルート・デバイスの接続を通知します．OTGドライバは，このルート・デバイスの接続通知を受けて，a_hostあるいはb_host状態へ遷移します．ホスト・ドライバが起動されてから規定時間内にこの通知がない場合，OTGドライバはホスト・ドライバを終了させステート・マシンを動作させます．

●ルート・デバイス切断

ホスト・ドライバはルート・デバイスの切断を検出したとき，OTGドライバへルート・デバイスの切断を通知します．OTGドライバは，このルート・デバイスの切断通知を受信すると，ホスト・ドライバを終了させてステート・マシンを動作させます．

●HNPイネーブル，ディセーブル処理

ホスト・ドライバは，ルート・デバイスのエニュメレーション時に接続されたルート・デバイスがHNPをサポートしているかどうかを確認します．HNPがサポートされているデバイスが接続されたときは，HNPをイネーブルにするSET_FEATUREリクエストをデバイスに発行します．同時にOTGドライバへHNPイネーブルの通知をします．ルート・デバイスがHNPをサポートしていないときは，HNPディセーブルをOTGドライバへ通知します．OTGドライバは，HNPイネーブル通知を受信している場合のみ，HNPの動作を許可します．

●レジューム検出

ホスト・ドライバはバスがサスペンドした状態からデバイス側からのレジューム信号を検出したときOTGドライバへレジューム検出を通知します．OTGドライバはレジューム検出通知を受信したら，バスをオペレーショナル状態へ遷移させます．

▶ ペリフェラル・ドライバ

ペリフェラル・ドライバは，ペリフェラル動作時にOTGドライバによって起動されます．ペリフェラル・ドライバは，ペリフェラル動作時は次のイベントが発生した際にOTGドライバへイベントの通知をします．

●コンフィグレーション

ペリフェラル・ドライバはSET_CONFIGURATIONを受信し構成ステートへ遷移したとき，OTGドライバへ構成ステートへの遷移を通知します．

●アン・コンフィグレーション

ペリフェラル・ドライバは構成ステートからほかのステートへ遷移したとき，OTGドライバへ構成ステートから外れたことを通知します．

●バス・サスペンド

ペリフェラル・ドライバはバスのサスペンドを検出したとき，OTGドライバへバスのサスペンド検出を通知します．

●バスレジューム

ペリフェラル・ドライバは，バスをレジュームする際にOTGドライバへ通知します．

●HNPイネーブル

ペリフェラル・ドライバは，HNPイネーブルのSET_FEATUREを受信したときにOTGドライバへHNPイネーブルを通知します．

● OTGマスストレージ・デモ・システムについて

次に，OTGマスストレージ・デモ・システムの構成，動作概要について説明します．

▶OTGマスストレージ・デモ・システムの構成

図13に，今回開発したOTGマスストレージ・デモ・システムのソフトウェア構成を示します．デモ・システムは，NORTi4上で動作します．

▶OTGマスストレージ・デモ・システム動作説明

OTGマスストレージ・デモ・システムは，接続形態によりホスト，ペリフェラルの機能を切り替えて動作します．次の三つの基本的な接続形態での動作を説明します．

●USBホスト機能

OTGマスストレージ・デモ・システムがUSBホストとして機能し，USBフラッシュ・メモリ，USBハードディスク，USB

メモリ・カード・リーダ/ライタなどのUSBマスストレージ・デバイスと接続し，ファイルのリード/ライトなどの操作を行うことができます〔図14(a)〕.

●USBペリフェラル機能

OTGマスストレージ・デモ・システムは，USBマスストレージ・デバイスとして機能し，ホスト・コンピュータと接続すると，リムーバブル・ディスクとしてPCに認識され，PC上からOTGマスストレージ・デモ・システム上のファイルの操作ができます〔図14(b)〕.

●OTG機能

2台のOTGマスストレージ・デモ・システムをUSBで接続して，データ通信を行うことができます．この場合，一方はUSBホストとして機能し，他方はUSBペリフェラルとして機能します．OTGのHNPによりホストとペリフェラルを入れ換えることが可能です〔図14(c)〕.

● デュアル・ロール・デバイスの開発手法

ここでは，OTGデュアル・ロール・デバイスを開発する際の構成方法，APIの使用方法，注意点などについて説明します．

▶デュアル・ロール・デバイスの構成方法

今回開発したOTGマスストレージ・デモ・システムはデュアル・ロール・デバイスであり，ホスト機能とペリフェラル機能を接続形態に応じて切り替えることのできるシステムです．デュアル・ロール・デバイスを開発するには，OTGドライバのほかに，次のようなUSB関連ドライバが必要です．

●USBホスト・ドライバ

USBホスト機能を制御するドライバです．USBデバイスの接続/切断の検出，USBデバイスのエニュメレーション処理，USBデバイスの管理などを行います．

●USBペリフェラル・コア・ドライバ

USBペリフェラル機能を制御するドライバです．USBホストからの標準リクエストの処理をします．

また，USBデバイス固有の機能を実現するために，ホストおよびペリフェラルのクラス・ドライバを組み込む必要があります．USB標準のクラス・ドライバを使用する場合には開発済み

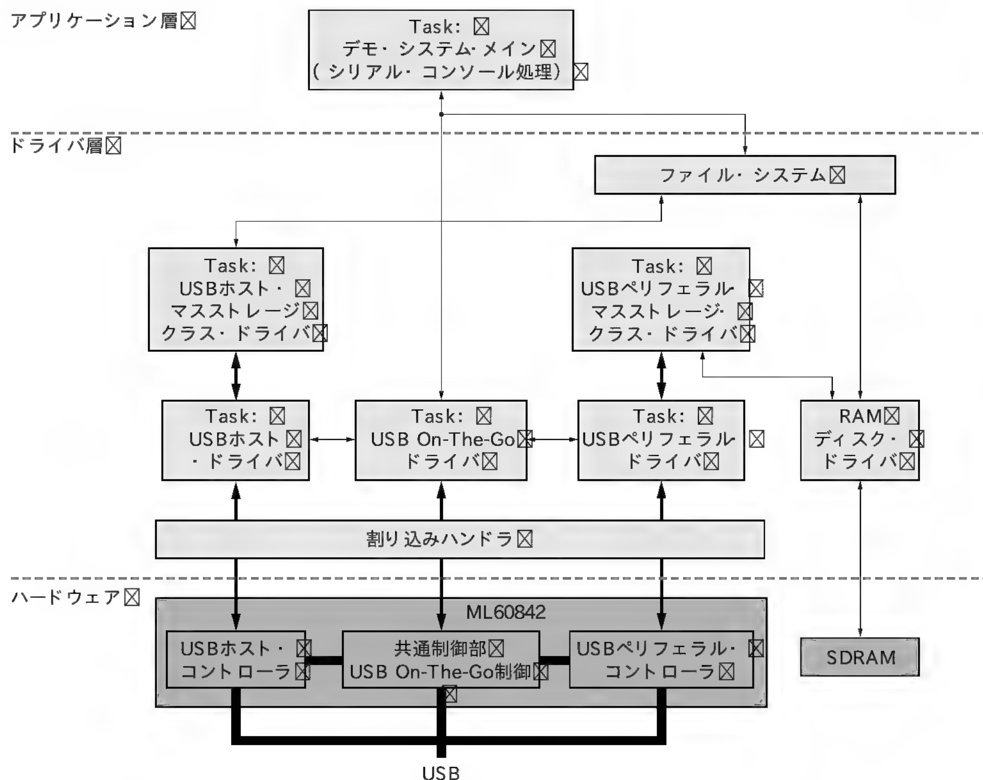


図13 OTGマスストレージ・デモ・システムのソフトウェア構成

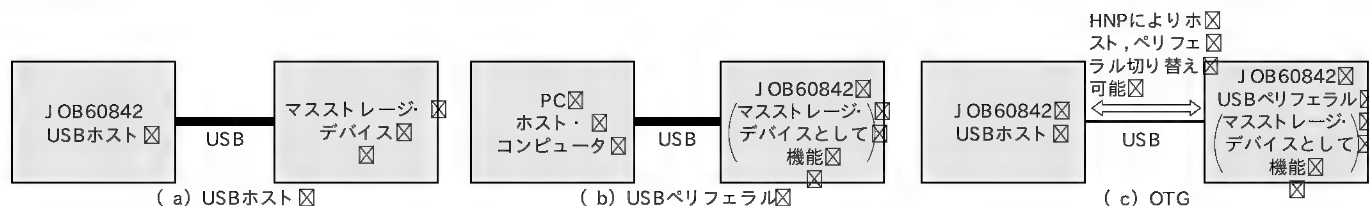


図14 各機能のデモ時の構成

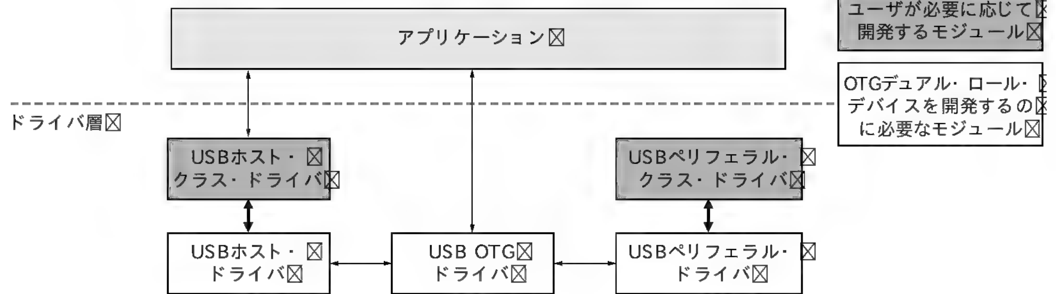


図15
デュアル・ロール・デバイスソフトウェア構成例

のドライバがあるので，それらを組み込んで使用できます．ベンダ固有のドライバを組み込む場合は，ユーザ側で開発する必要があります．

さらに，デュアル・ロール・デバイスを制御するためのアプリケーションも，ユーザで開発します．これらのソフトウェアを組み合わせることにより，容易にデュアル・ロール・デバイス・システムを実現することができます．

図15に，デュアル・ロール・デバイスのソフトウェアの構成例を示します．

▶ デュアル・ロール・デバイスのアプリケーション

OTGドライバをはじめとしたUSB関連のドライバは，すべてNORTi4上で動作します．したがって，デュアル・ロール・デバイスのアプリケーションは一つ以上のタスクで構成する必要があります．

OTGドライバに対しては，次のAPIを使用してリクエストを発行します．

関数名：usbctrl_otg_request
書　式：void usbctrl_otg_request(int request)
引き数：requesut
　　遷移条件設定要求の種類を表すパラメータ

アプリケーションは，OTGドライバに対して，次のリクエストを使用してデュアル・ロール・デバイスを制御します．

● USBバス使用要求（BUS_REQ）

USBのバスの使用要求をOTGドライバへ送信します．Aデバイス状態のとき，OTGドライバはホスト・ドライバを起動して，USBホストとなります．Bデバイス状態のとき，SRPを実行し，HNPが実行されるのを待ち，HNPが成立した後，USBホストとなります．

● USBバス・サスペンド要求（SUS_REQ）

USBのバスのサスペンド要求をOTGドライバへ送信します．Aデバイス状態のとき，OTGドライバはHNPを実行します．HNPに成功するとUSBペリフェラルとなります．Bデバイス状態でUSBホストとして動作しているときは，バスを開放して，アイドル状態に戻ります．

● USBバス開放要求（DROP_REQ）

USBバスの開放要求をOTGドライバへ送信します．Aデバイス状態のとき，OTGドライバはホスト・ドライバを終了させ，アイドル状態へ戻ります．

アプリケーションは，ホスト動作時はクラス・ドライバが提供するAPIを使用して接続されているペリフェラル機器にアクセスします．ペリフェラル動作時は，ペリフェラル・クラス・ドライバでPCからのアクセスに対して処理を行うので，アプリケーションではペリフェラル・クラス・ドライバの動作を妨げるような処理を行ってはいけません．また，アプリケーション・タスクの優先度は，各USBドライバの優先度よりも低く設定する必要があります．

まとめ

現在，USBはPCと周辺機器を接続するインタフェースとして主流となっています．しかし，USB On-The-Go Supplementは2001年12月に制定されてから2年半が経ちますが，OTGの利点を生かしたアプリケーションがないため，まだ本格的に市場に普及していません．

OTGデバイスには，ペリフェラルとホストの両機能が必要です．OHCIなどホスト制御は複雑ですが，OTGのホスト機能は“限定されたホスト”で十分であり，接続を必要とする機器を限定することで，開発は容易になります．オーディオ・プレーヤや携帯情報端末（PDA）がOTG機能をもつことで，PCとしか接続できなかったUSBが，周辺機器どうしで接続できるようになり，応用範囲が広がります．ぜひ，OTGシステムの開発を検討してみてください．

みやた・まなぶ/おかざき・しんや/さいとう・たかゆき　沖電気工業（株）

DOSベースで動作するサンプル・プログラムによりOTG制御を容易に理解できる

6 ISP1362の概要とOn-The-Go サンプル・プログラムの詳細

岡野 彰文

もう一つ，On-The-Go対応デバイスとしてISP1362を取り上げる．PCIバス・ベースの評価ボードと，DOSベースで動作するサンプル・プログラムが公開されているので，On-The-Goの制御プログラムの実例を容易にテストすることができる．ここではISP1362について解説した後，このサンプル・プログラムについても詳しく解説する　　（編集部）

はじめに

USBはPCのインターフェースとして，今やどのプラットホームにおいても標準的なものとなり，その応用範囲は対応デバイスの増加とともにますます拡大し，また低価格化が進んでいます．一方，注目のディジタル家電の世界でもこのような市場の風をうまく利用し，低価格で手軽な機器間インターフェースとしてUSBがすでに多く採用され，標準インターフェースといえる地位を確立しつつあります．

しかし，組み込み系，特にディジタル家電のようなアプリケーションでのUSBは，ターゲット側（スレーブ，ファンクション，デバイスなどとも呼ばれる）機能の実装が多く，ホスト側（マスタ側）を搭載するアプリケーションはあまりありません．

USBホストを実装するには，通常のホストとして用意する方法と，On-The-Go（OTG）規格準拠として用意する方法があります．OTGは2001年末にUSB2.0のサプリメントとして策定され，2003年6月にはRevision1.0aに改訂されています．

通常のホストとして実装する場合は，ホスト専用のUSBコネクタが必要になります．OTGに準拠すると，ホストとターゲットのコネクタを共有させることできます．

Philips社の組み込み機器向けUSBホスト・コントローラ・シリーズは，このような組み込み機器で，高パフォーマンスのUSBホストを容易に実装するために開発されました．

すでにプリンタやMP3プレーヤなどでは，USBホスト機能を備えている機器が多くあります．これらのアプリケーションでは，Philips社の組み込み向けホスト・コントローラ・チップはポピュラな存在となっています．

1 Philips社のUSBコントローラの概要

● ホスト・コントローラ製品構成

Philips社のUSBホスト・コントローラ製品群は表1のようなシリーズ構成になっています．これらのチップは，すべてアナログ・トランシーバを内蔵しており，1チップでUSBホスト（さらにデバイスやOTG）に必要な機能が実現できます．図1に製品群のロード・マップを示します．

表1 USBホスト・コントローラ製品群

品　名	スピード	内蔵コントローラ	USBポート	システム・バス・インターフェース	内蔵RAM
ISP1161A1	フル/ロー・スピード	ホスト/ターゲット	ホスト×2 デバイス×1	CPU汎用バス	4Kバイト（ホスト） 2462バイト（ターゲット）
ISP1160/01	フル/ロー・スピード	ホスト	ホスト×2	CPU汎用バス	4Kバイト（ホスト）
ISP1362	フル/ロー・スピード	ホスト/ターゲット/OTG	OTG×1 注A ホスト×1	CPU汎用バス	4Kバイト（ホスト） 2462バイト（ターゲット）
ISP1261	フル/ロー・スピード	ホスト/ターゲット/OTG	OTG×1 注A	USB 注B	4Kバイト （ホスト/ターゲット共用）
ISP1561	ハイ/フル/ロー・スピード	ホスト	ホスト×4	PCI	―
ISP1760	ハイ/フル/ロー・スピード	ホスト	ホスト×3	CPU汎用バス	64Kバイト（ホスト）
ISP1761	ハイ/フル/ロー・スピード	ホスト/ターゲット/OTG	OTG×1 注A ホスト×2	CPU汎用バス	64Kバイト（ホスト） 8Kバイト（ターゲット）

注A：OTGポートは設定によりホストまたはターゲット・ポートとして固定可能．
注B：ISP1261のシステム接続はUSBを介して行う．ISP1261はシステム側専用のUSBダウン・ストリーム・ポートを持つ．

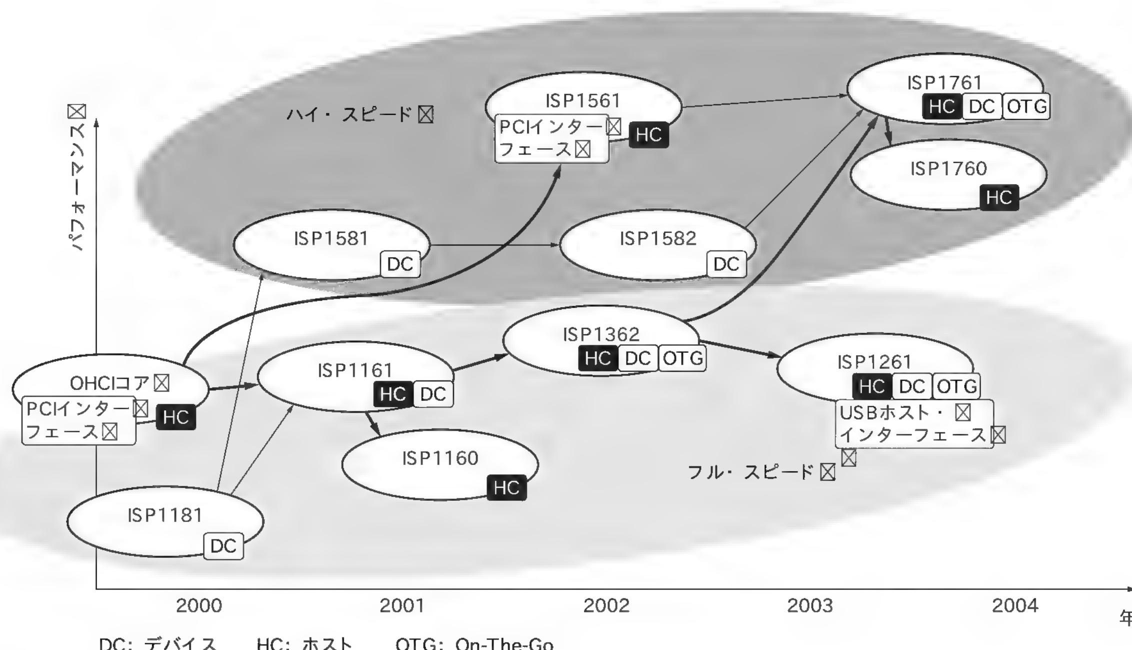


図1　USBホスト・コントローラ製品のロード・マップ

● 組み込みシステム向けホスト・コントローラの特徴

「組み込み向け」のホスト・コントローラとはどのような特徴を持ったチップなのでしょうか．

EHCIやOHCI，UHCIのような汎用のホスト・コントローラ・チップは，システム側バスとしてPCIバスを使うことを前提としています．PCIバスを持たないターゲット・システムに，このようなチップを使おうとする場合，PCIバス・ブリッジのような余計なハードウェアが必要となってしまいます．

また，PCIブリッジを使ったにせよ，実際にこれらのホスト・コントローラが動作するためには，そのシステム・バスが，周辺チップに対してマスタになることを許可するしくみがなくてはなりません．これは汎用ホスト・コントローラ・チップとCPUが，共有するメモリを相互にアクセスして，通信を実現する構造を持っているためです．

一般に組み込みシステムの場合，このようなバスを用意するのは困難でしょう．多くの場合，「CPU搭載チップだけがマスタになれる」ようなバスとなっています．これが組み込みシステムにおいて汎用ホスト・コントローラ・チップの採用が難しい理由です．

組み込み用USBホスト・コントローラ・チップはこれらの制限をなくし，組み込みシステム上でUSBホストを簡単に実現するために開発されました．システム側のバスにSRAMに類似した汎用バス・インターフェースを採用し，多くのCPUチップと容易に直接接続できます．また，通信に使用するバッファ・メモリをチップに内蔵しています．この内蔵メモリを使用することにより，ホスト・コントローラはみずから外部メモリにアクセスを行う必要がなくなり，システム・バスに対してつねにスレーブとしての動作で通信が行えます．組み込み向けホスト・コントローラ各チップのデータ・シートには「USB Slave Host Controller」などの記述がありますが，このSlaveとは上記のような特徴を意味しています．

このようなシステム構成の上に，さらに独自の制御機構を採用することで，CPUやバスに対するUSB通信時のシステムの負荷の低減を実現しています．

2 OTGコントローラISP1362の概要

この項では，組み込みシステム向けUSBホスト・コントローラの，より具体的な実装の例として，現在量産中で，市場での実績も多いISP1362（Philips社製）を取り上げます．まず，チップ自体についての解説を行い，その後，評価キット/サンプル・コードを用いて，アプリケーションの例やソフトウェアの実現方法を解説します．

ISP1362は一つのチップにホスト，ターゲット，USB On-The-Go（以下OTG）の各機能を集積したチップです．ホストとターゲットの各ブロックは独立しており，両方の機能を同時に使うことが可能です．

パッケージは10mm角のQFPと6mm角のBGAの2種類（どちらも64ピン）が用意されています．

● ブロック構成

写真1にパッケージ外観を，図2にISP1362の内部ブロックを示します．

▶USBポート

ISP1362には2個のUSBポートが用意されています．各ポー

12MHz
CLKOUT
RESET
パワーON
リセット
HCバッファ・
メモリ
PLL
システム・クロック
H_SUSPEND/
H_WAKUP
ホスト・コントローラ
(HC)
D0～D15
On-The-Go
コントローラ
RD
CS
WR
A0
A1
DACK1
DACK2
DREQ1
DREQ2
INT1
INT2
バス・
インター
フェース
ターゲット(デバイス)
コントローラ
(DC)
DCバッファ・
メモリ
GOODLINK
過電流
保護
USB
トランシーバ
OTG
トランシーバ
チャージ・
ポンプ
V_{DD} 5V
H_PSW1
H_PSW2
H_OC1
H_OC2
H_DM2
H_DP2
OTG_DM1
OTG_DP1
V_{BUS}
DGND
AGND
V_{CC}
D_SUSPEND/
D_WAKEUP
GL
OTGMODE
ID
CP_CAP2
CP_CAP1

図2 ISP1362の内部ブロック

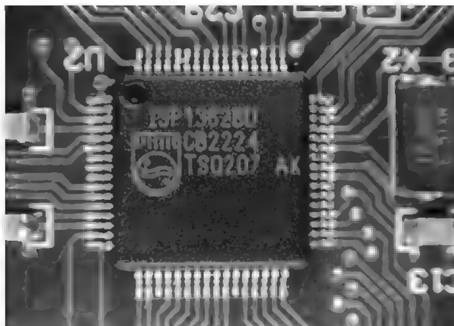

写真1
ISP1362の外観

表2
ISP1362のUSBポートの構成

	ポート1	ポート2
構成1	OTG	ホスト
構成2	ホスト	ホスト
構成3	ターゲット	ホスト

トは表2のいずれかの組み合わせに設定することができます．

ポート1はOTGポートとして使うか，あるいはホスト，デバイスのどちらかに固定して使うことができます．ポート2はホスト専用のポートで，もし必要がなければこの機能を停止させておくことも可能です．

図3はより簡略化したチップのブロック図で，図下側に書かれている各USBポートが内部ブロックにどのように接続されているかを示しています．ポート2はホスト・コントローラ下のルート・ハブに直接接続されており，ホスト固定ポートとして使われます．もう一方のポート1はルート・ハブとの間にスイッチが設けられ，これを必要に応じて切り替えることができます．

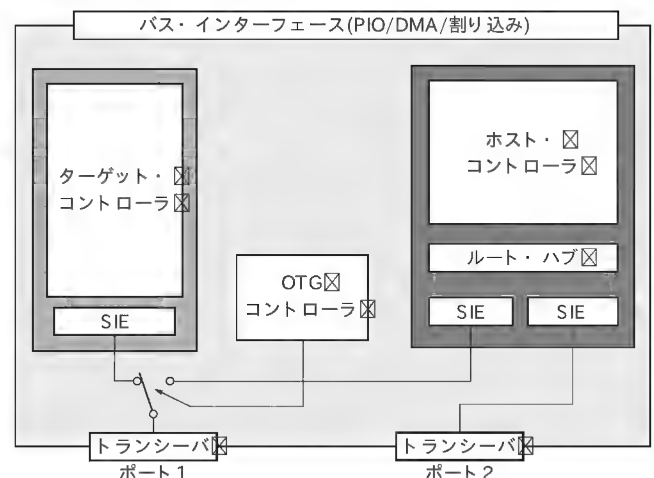


図3 ポート1/2とホスト/ターゲット・コントローラの関係

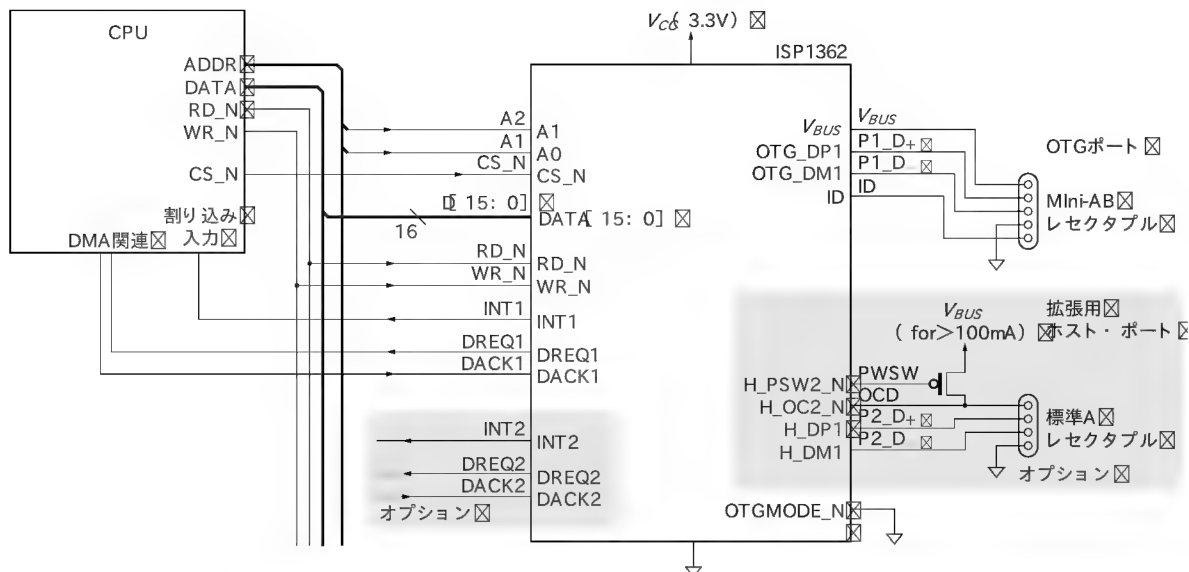


図4　組み込み用CPUとISP1362の接続回路例

ポート1はOTGのポートとして使用するか，あるいはホストまたはターゲット，どちらかのポートとして固定的に使うことができます．

▶ システム・バス

一般的な組み込み用CPUとISP1362の接続回路例を図4に示します．

CPU(システム)側のバス・インターフェースは16ビット幅のデータ・バスと2本のアドレス線，リード/ライト線，チップ・セレクト線となっています(バス・タイミングの詳細はデータ・シートの第20章を参照)．これに加えて内部ホスト，ターゲットのブロックから各1本の割り込み，各1チャネルのDMA(DREQ，DACK)がチップの外部に出ています．これら2本の割り込み線，2チャネルのDMAは内部レジスタの設定により，1本の割り込み，1チャネルのDMAにまとめることができます．

● 電源/USBバス電源(V_{BUS})

ISP1362は3.3Vの単一電源で動作します．CPU側システム・バスの各ピンは5V信号入力に対応しています．チップの非動作状態(ホスト/ターゲットともサスペンド)時にクロック発振を停止させ，超低消費電力状態にすることも可能です．

USBホスト機器はUSBコネクタから，5VのUSBバス電源(V_{BUS})を供給しなければなりません．システムにはこのための電源が必要になりますが，ISP1362ではこれを内部のチャージ・ポンプで3.3Vを昇圧し，供給することができます[注1]．もし，すでにシステム内に5V電源が用意されている場合には，このチャージポンプの動作を停止させ，別に用意された5V電源をV_{BUS}に用いることが可能です．

V_{BUS}に外部電源を用いる場合，内蔵の過電流検出回路を使うことができます．これによりごく簡単な外付けのMOS-FETをスイッチとして付加するだけの回路で，過電流保護機能を実現できます．

● ホスト・コントローラ部

▶OHCIコア部とバッファ管理部

ISP1362の内蔵ホスト・コントローラは，OHCIコア部分とバッファ管理部分で構成されています．OHCI部分は，PCなどで使われる汎用のOHCIコントローラと同じコアが使われています．バッファ管理部分は，このOHCIコアを組み込みシステムで使うために設けられたラッパ部と考えることができます．

ソフトウェアから見ると，内部のレジスタ・コマンドの`0x00`から`0x16`までにマップされたレジスタ群が，OHCIコアに属する部分となります．そのほかの部分はISP1362のハードウェアとバッファ管理に属する部分です．図5にレジスタの概要を示します．レジスタの一覧ならびに詳細は，ISP1362データ・シートの第15章を参照ください．

▶ 内蔵メモリ

ホスト部には4Kバイトの内蔵メモリが用意されています．すべての転送はこのメモリを用いて管理されます．メモリはユーザの設定によってISTL，INTL，ATLと呼ばれる，任意のサイズの三つのブロックに分けられます．それぞれのブロックでアイソクロナス転送，インタラプト転送，非周期転送(コントロール/バルク転送)を管理します〔図6(a)〕．

転送はエンドポイントごとにトランスファ単位で管理できます．この管理には，メモリ中に置かれるPhilips Transfer

注1：このチャージ・ポンプが供給できる上限は20mAまで(OTGには規定により8mA以上を供給できる電源が必要)．OTGではなくホストのフル規格を満たす場合には100mAあるいは500mA以上を供給できる電源が別途必要になる．

ISP1362

ホスト・コントローラ
A[1:0]=0X'

OHCI
HcRevision
HcControl
HcCommandStatus
HcInterruptStatus
HcInterruptDisable
HcInterruptEnable
HcFmInterval
HcFmRemaining
HcFmNumber
HcLSThreshold

ルート・ハブ
HcRhDescriptorA
HcRhDescriptorB
HcRhStatus
HcRhPortStatus[1]
HcRhPortStatus[2]

INT1
INT2
DREQ1
DREQ2
DACK1
DACK2

HcmPInterrupt
HcmPInterruptEnable
HcChipID
HcScratch
HcSoftwareReset
HcDMAConfiguration
HcTransferCounter
OneINT
HcHardwareConfiguration
OneDMA

HcBufferStatus
HcDirectAddressLength
HcDirectAddressData

HcISTLBufferSize
HcISTL0BufferPort
HcISTL1BufferPort
HcISTLToggleRate
HcINTLBufferSize
HcINTLBufferPort
HcINTLBlkSize
HcINTLPTDDoneMap
HcINTLPTDSkipMap
HcINTLLastPTD
HcINTLCurrentActivePTD
HcATLBufferSize
HcATLBufferPort
HcATLBlkSize
HcATLPTDDoneMap
HcATLPTDSkipMap
HcATLLastPTD
HcATLCurrentActivePTD
HcATLPTDDoneThresholdCount
HcATLPTDDoneThresholdTimeout
バッファ・コントロール

OTGコントローラ
A[1:0]=0X'
OtgControl
OtgStatus
OtgInterrupt
OtgInterruptEnable
OtgTimer
OtgAltTimer

デバイス・コントローラ A[1:0]=1X'
DcMode
DcHardwareConfiguration
DcInterrupt
DcInterruptEnable
DcScratch
DcChipID
DcAddress
DcFrameNumber
DcDMAConfiguration
DcDMACounter
DcEndpointConfiguration
DcEndpointStatus
DcEndpointStatusImage
DcErrorCode

図5 ISP1362のレジスタ構成

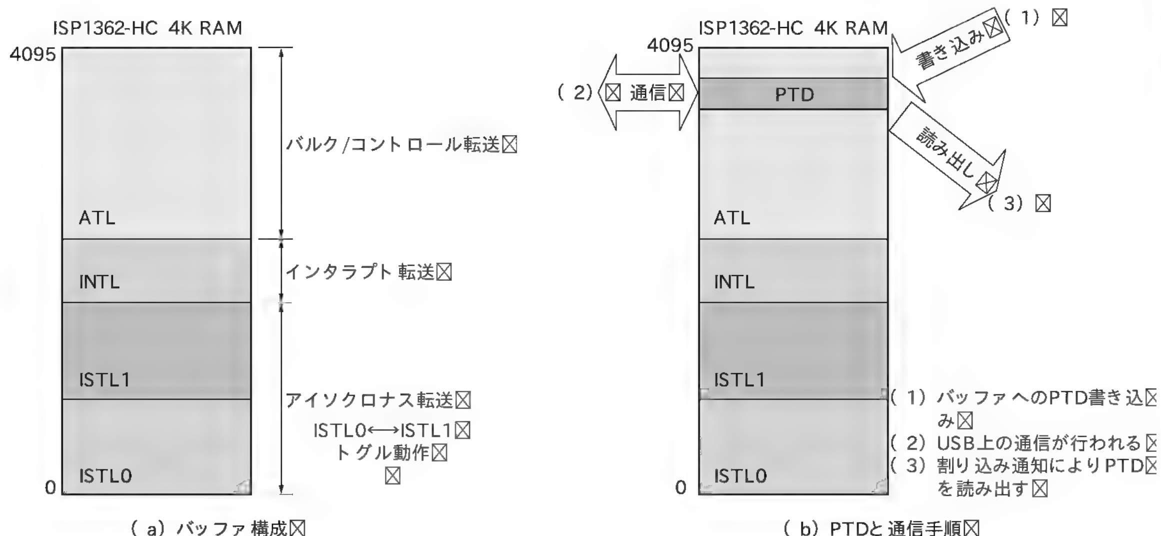


図6
内蔵メモリのバッファ構成

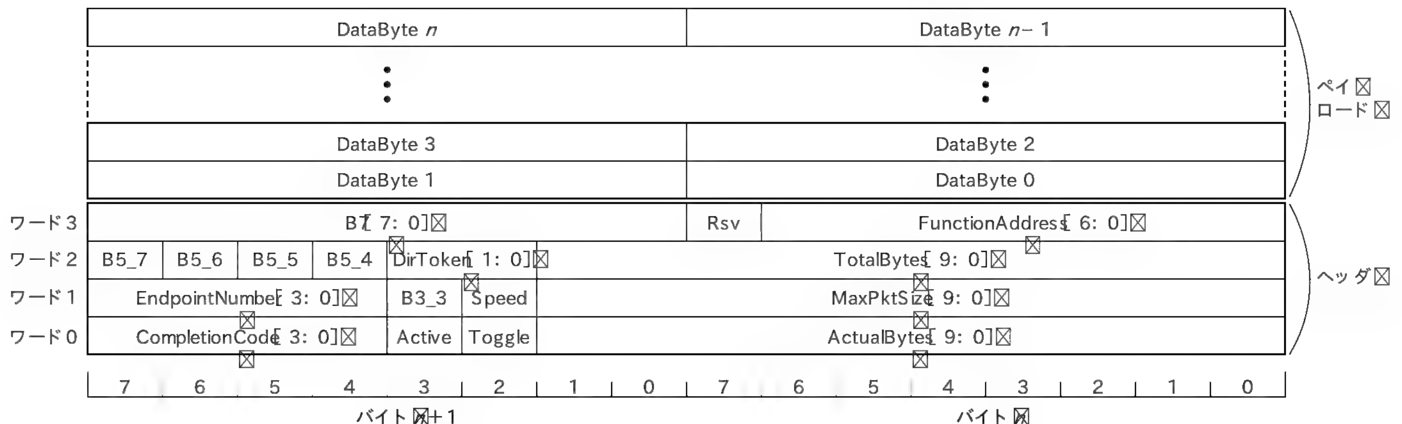


図7　PTDの構成

Descripter(PTD)と呼ばれる，データ構造が使われます．PTDは8バイトのヘッダと転送データを収納するペイロード部から構成されます(図7)．CPUはこのPTDをチップ内のバッファ・メモリに書き込み，転送終了後PTDを回収する作業を繰り返して転送を実現します．

PTDヘッダには転送に必要な付帯情報が格納されます．ターゲット・デバイスのアドレス，エンドポイント番号，エンドポイントのサイズ，転送のスピード，転送方向，データ・トグルの指定，ペイロードのサイズ，また，転送終了時にその結果をレポートするCompletionCode，転送の行われた実際のデータ量が反映されるフィールドなどがあります(PTDの詳細はデータ・シートの第12.4節を参照)．

PTDペイロード部は，最大1023バイトのデータを格納できます．いったんPTDをセットすると，転送はISP1362によって自動的に行われ，その完了が割り込みによって通知されてPTDの読み出しを要求するまでの間，CPUはUSB以外のほかの仕事に時間を割くことができます[図6(b)]．

バッファ・メモリには同時に複数のPTDをセットすることができます．ISP1362ではハードウェアのレベルで，インタラプト転送，非周期転送のそれぞれに，最大32個のPTDを同時に設定し，各エンドポイントをターゲットとした転送をいっしょに，自動的に行わせることが可能です．スケジューリングはハードウェアによって行われます．アイソクロナス転送では，ターゲットとするエンドポイントの数に制限はありません．

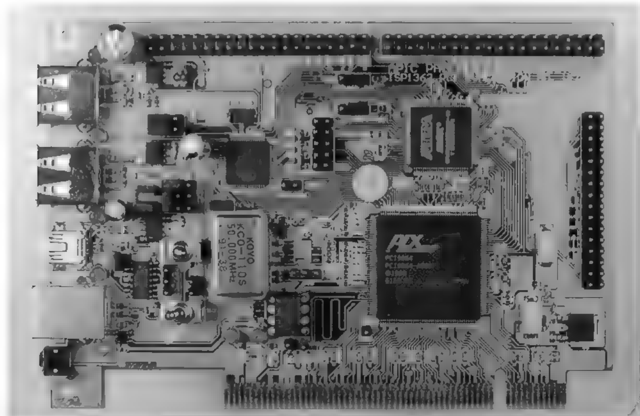


写真2　ISP1362評価キットの外観

● ターゲット・コントローラ部

ターゲット・コントローラはフル・スピード・デバイス・コントローラISP1181(Philips社製)と同一のコアが内蔵されています．このターゲット・コントローラは，コントロール・エンドポイントのほかに，14本までの任意のサイズのエンドポイントを使うことができます．ターゲット・コントローラの制御ソフトウェアには，このISP1181と同一のものが使えます．

● OTGコントローラ部

OTGコントローラ部には，OTG機能を実現するため，USBのデータ線，V_{BUS}の制御とモニタを行うためのスイッチ，アナログ・コンパレータ類が内蔵されており，これらに対応したレジスタ，さらに割り込みを制御するレジスタ群が用意されています．

3 ISP1362評価キットの概要

ISP1362評価キットは，同チップの標準評価システムです(写真2)．このキットには評価ボード，評価ソフトウェア，さらにマニュアル，アプリケーション・ノート，回路図などのドキュメントが含まれています．

● 評価キット・ハードウェア

ISP1362評価ボードは簡単にチップの評価が行えるよう，手軽にPCで利用できるPCI評価ボードとソフトウェアのセットで供給されています．

この評価ボードでは，PCIバスとISP1362バスをPCIブリッジ・チップで接続し，ISP1362レジスタがPCのI/O空間にマップされるように設定されています．これにより，PC上のソフトウェアはISP1362のレジスタに直接アクセス，操作を行うこ

とができます．

また，ISP1362につながるローカル・バスの各信号線は，基板上の端子に用意されています．この端子と組み込みターゲット・システムのバスを接続し，実機での評価も行えるようになっています．

基板上にはUSBポートとして標準A，標準B，Mini-ABコネクタが用意されており，ボード上に用意されたスイッチ，ジャンパ・ピンによって使用方法を選択できます(ポート2は標準Aコネクタに固定となっている)．

● 評価キット・ソフトウェア

評価キットには，このボードを動作させるサンプル・ソフトウェアがバンドルされています．評価キットは，バンドルされるソフトウェアの種類によって，Linuxキット，Windows CEキット，DOSキットの3種類が用意されています．

Linux，Windows CEキットは，それぞれのOSに対応したISP1362に必要なソフトウェア・ドライバが付属し，各OSプラットホームにインストールして評価できるようになっています(これらのドライバはHCD部[注2]のみとなる)．もし，開発ターゲット・システムにLinuxやWindows CEが使われるのであれば，このコードの移植で，ターゲットにUSBホスト機能を実現できます．

このほか，ISP1362のドライバを用意するには既成の商用ドライバを用いる方法もあります．Philips社からはFlexistackと呼ばれる，HCDからクラス・ドライバまでの層を含んだドライバ・スタックが販売されています．各種リアルタイムOSやCPUに容易に移植できる構造を持っており，ほとんどあらゆる種類のターゲット・システムに対応可能です．また，このほかにもISP1362に対応したUSBホスト・ドライバが各社から販売されています．

さらにもう一つの選択肢として，必要なソフトウェアをユーザ自身で作成することも可能でしょう．ISP1362の詳細はデータ・シートで公開されており，プログラミング・ガイドや，そのほかサンプル・コードも用意されています．DOSキットはこのようなソフトを自身で作成するユーザを対象として販売されています．

4 サンプル・プログラム WASABI-Hot!の詳細

DOSキットにはサンプル・コードとしてWASABI-Hot!プログラムが付属します．ここではこれを例にコーディングの実際を解説します．

● WASABI-Hot!の目的と概要

WASABI-Hot(以下WASABI)は，ISP1362の評価ボードを用いたチップの動作検証とデモンストレーションを行うためのサンプル・コードとして開発されました．ISP1362の基本的な機能を網羅し，OTG動作やホスト，デバイスとして各種転送を実行できます．図8に実行画面の例を，図9に典型的なデモのセットアップ例を示します．

このサンプル/デモ・コードはアプリケーション単体で動作し，実行ファイル以外のほかのドライバなどを必要としません．実行ファイルをコピーし，立ち上げるだけで動作します．チップへのアクセスはアプリケーション内から直接，I/Oポートへのアクセスとして行います[注3]．

このような単体での動作は，組み込み向けのサンプル・コードとしては非常にわかりやすいと思います．LinuxやWindows



図8 実行画面の例

注2：ホスト・ドライバの構造については後述する「一般的なホスト・ドライバ・スタックとの構造の比較」の節を参照．

注3：アプリケーションからハードウェアへのアクセスを直接行うため，DOSか16ビット系Windowsの環境で動作させること．32ビット系Windowsでは動作しない．

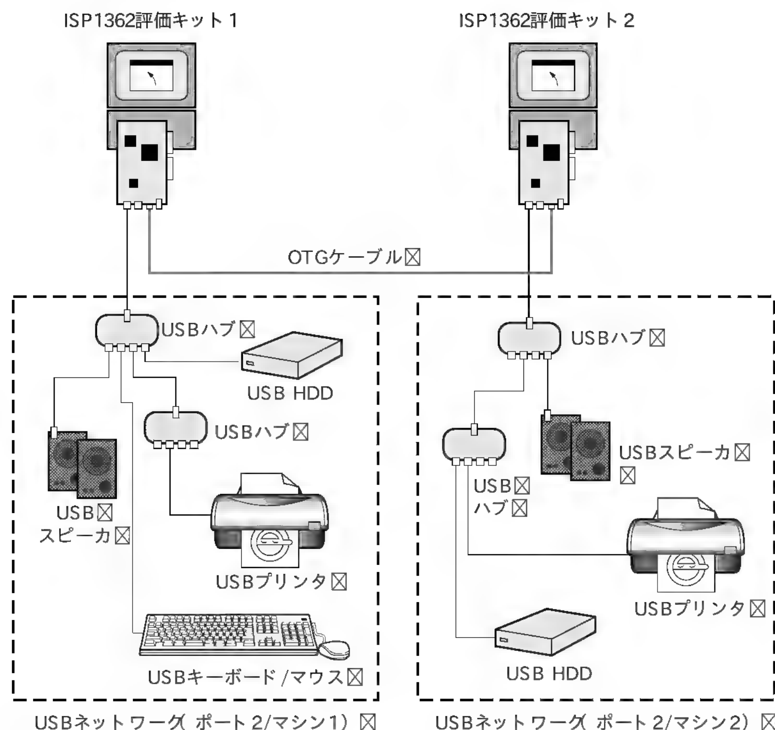


図9　デモのセットアップ例

CE向けのサンプル・コードでは，USBのバス管理などのUSB動作の基本部分はOSが提供するため，ハードウェアに依存する部分(HCD)だけがドライバとして提供されています．これに対し，WASABIはそれ単体でUSBホスト動作をするためのすべての部分をコード中に含んでいます．このため，たとえばWASABIを移植することができれば，USB非対応であったプラットホームにUSBホスト機能を実現することが可能です[注4]．

またWASABIは，ISP1362が少ないソフトウェア資源で動作可能であることを示す良い例であるといえます．WASABIは単純なDOSアプリケーションであり，リアルタイムOSのようなソフトウェアの実行管理を行う機能を用いず，USBホスト，さらにOTGに必要な機能を実現しています．

● 配布条件

WASABIはISP1362-DOS評価キットの一部として供給されるソフトです．いわゆるフリー・ソフトウェアであり，サポートがないこと，いかなる保証もないこと，さらにこのソフトがPhilips社のチップにのみ使用されることを前提に，ソース・コード，実行バイナリを自由に使うことができます．コードの改変，ほかのプラットホーム(CPU，OS)への移植，そのほかユーザの使用に特に制限はありません．

コードのほかに，ユーザ・マニュアル(pdf)，ユーザ・ガイド(html)が配布目的で用意されています．ユーザ・ガイドではおもにデモの概要を，ユーザ・マニュアルではより詳しいデモの方法と，内部構造の概要，既知の問題点などが解説されています．

● 機能詳細

WASABIはデモ用コードとしての多くの機能が用意されています．ここではホストとしての動作の観点から，サポートしているデバイスの紹介とその動作について解説します．ここではこれらの概要を紹介するにとどめます．詳細や操作方法についてはユーザ・マニュアルを参照ください[注5]．

▶USBハブ

チップの解説の節で述べたように，ISP1362には2ポートのルート・ハブが内蔵されています．このルート・ハブはISP1362内部のレジスタ操作で直接制御できます．もし，外部ハブをサポートしなくてよいシステムを実現する場合には，あえてハブ・クラス・ドライバを実装せず，ホスト・コントローラのドライバ内部でレジスタを直接操作し，ソフトウェアを簡略化することが可能です．

WASABIではより多くのデバイスを一度に接続できるよう，「(通常の)外部に接続されるハブ」をサポートしています．ハブ制御のためのソフトウェアはハブ・クラス・ドライバとして用意されています．このドライバはルート・ハブ，外部ハブを一元的に管理します．

▶プリンタ

簡易プリンタ・ドライバが用意されています．このドライバは，あらかじめPC上のプリンタ・ドライバを用いて作成した，プリント・イメージ・ファイルを用いてプリントを実行します．つまり，イメージ・データの変換は行わず，単純に用意されたファイル中のデータをプリンタの指定されたエンドポイントに転送することにより，そのプリント作業を実行します．一般的なインクジェット方式のプリンタでの動作が可能です．

WASABIでは複数のエンドポイントを制御する例として，プリンタを複数台同時にドライブすることが可能です．

▶USBスピーカ

アイソクロナス転送デモとして，USBスピーカを鳴らすことができます．アプリケーション内で作られる波形データをビープ音として，あるいは，あらかじめPC上のHDDに保存されたWAVファイルをUSB経由でスピーカに転送して演奏します．

▶キーボード/マウス

Human Interface Device(HID)としてキーボードとマウスをサポートしています．WASABIのアプリケーション内でUSBのキー操作を受け付けることができ，マウス操作を行うと，その動作を画面に反映するように設定されています．

注4：ただし，WASABIはいわゆるHackである．コードはデモの実現を第一目標として書かれているため，その動作，内容の妥当性は保証の範囲外．
注5：この節以降，WASABIのコードの説明として使われる「ドライバ」ということばは，アプリケーション・コード内部のモジュールを指す．これらはアプリケーションの一部として用意され，実行される．

▶ ストレージ・デバイス

データ高速転送のデモ機能として，ストレージ・クラス・ドライバが用意されています．ファイル・システムのハンドラまでのモジュールを用意し，USBに接続されるストレージの処理すべてをWASABIの内部で管理しています．コマンドにSCSI，プロトコルにバルク・オンリ転送，ファイル・システムにFAT12/FAT16を用いるストレージ・デバイスをドライブできます．

このデモでは，一般的なファイル・システムのアクセス機能と，ファイル転送が行えます．

● 配布パッケージの詳細

WASABIは，フォルダ/ファイルを含むフォルダの(あるいはその圧縮された)形で配布されています．それぞれのファイル/フォルダは表3のように構成されています．

● ソース・コードの詳細

先にも述べたようにWASABIは「簡単なセットアップでISP1362のデモを行う」ために作成されました．この目的を実現するため，実行前にほかのファイルやドライバをインストールすることなく実行でき，その単一アプリケーションの内部でUSBの基本的なホスト機能のすべてを実行します．

また，このオールイン・ワン構成の利点を生かし，USBホスト非対応のほかのプラットホームへの移植が行われることも念頭に設計されています．高い移植性を提供するために，すべてのコードはANSI-C準拠のC言語で書かれており，特別なライブラリやドライバを必要とせず，また標準ライブラリも極力使用しない方針で書かれています．

しかし例外として，たとえばスクリーン描画，ローカル・ファイル・アクセス，割り込みハンドリング，レジスタ・アクセスのためのコールなどはプラットホームに依存したものをやむをえず使用しています．ただし，これらのコールはすべて一定の抽象化レベル以下で用いられています．

配布されているソース・コードはTurbo-C 3.0でビルドできます．ソース・コードは最小限の変更で他の開発ツールでビルドすることも可能です．

▶ ユニットとディレクトリの構成

ソース・コード(モジュール)は機能別にサブディレクトリにまとめられています．ここではこのモジュールのまとまりを便宜的に「ユニット」と呼ぶことにします．表4に各ユニットの機能をまとめます．

▶ クラス・ドライバ

クラス・ドライバとして5種類のサンプルが用意されています(表5)．

● プログラムの構造

▶ アプリケーションの実行/メイン・ループ

WASABIは単純なDOSアプリケーションであり，その内部はマルチタスクやマルチスレッドのような，並列実行機構を用いずに実現されています．USBホストのごく簡単な実装，たとえば同時に転送の対象を一つのエンドポイントに制限したようなシステムでは，このような制限はまったく問題となりませんが，WASABIの欲張った仕様では，これに反し複数の仕事を並列に実行できなければなりませんでした．

たとえば，ポート1でOTGまたはターゲットとしての動作をしながら，ポート2はホストの仕事を継続するような場合，この両方をメンテナンスする必要があります．さらにストレージが接続されている場合には，これのめんどうを見る処理を継続してやらねばなりません．

WSABIでは，並列処理を疑似的に実現しています．あらかじめ特定の関数を(WASABIでは「メンテナンス・ルーチン」と呼ぶ)を登録しておき，この関数をアプリケーション・コードの各所に配置された「待ち」の時間にコール，実行するような仕掛けを作ってあります．この構造は不用意な関数の再帰呼び出しを行う可能性があることと，実行に要求されるスタック・サイズを把握しにくくなるという問題がありますが，簡易的な実装としては目的を達成しています．

実際にRTOSを用いるようなシステムに移植する場合には，このメンテナンス・ルーチンはタスクやスレッドとして実装す

表3 WASABI配布ファイルの内容

WASABI*/readme.txt	説明ファイル
WASABI*/guide.html	簡易オンライン・ガイド
WASABI*/obj/	アプリケーションの実行コード本体
WASABI*/src/	ソース・コード・ディレクトリ
WASABI*/WASABI-h.prj	Turbo-Cプロジェクト・ファイル
WASABI*/setting.wsb	アプリケーション実行のための設定ファイル
WASABI*/song.lst	USBスピーカ使用時のWAVEファイル・パス・リスト
WASABI*/class_dr.txt	カスタム・クラス・ドライバ作成時の説明

表4 各ユニットとディレクトリの構成

WASABI*/src/	アプリケーション・ユニット
WASABI*/src/_hc_core/	ホスト・コア・ユニット
WASABI*/src/_hc_hw/	ホスト・ハードウェア・ユニット
WASABI*/src/_hc_cls/	クラス・ドライバ管理ユニット
WASABI*/src/class_dr/	クラス・ドライバ
WASABI*/src/_dc/	デバイス・ユニット
WASABI*/src/_otg/	OTGユニット

表5 クラス・ドライバ

WASABI*/src/class_dr/hub/	ハブ・クラス・ドライバ
WASABI*/src/class_dr/hid/	HIDクラス・ドライバ
WASABI*/src/class_dr/audio/	USBスピーカ(アイソクロナス転送)デモ
WASABI*/src/class_dr/printer/	プリンタ印刷(バルク転送)デモ
WASABI*/src/class_dr/storage/	マスストレージ・クラス・ドライバ，ファイル・システム・ハンドラ(FAT16，FAT12)および簡易コマンド・シェル

るほうが良いでしょう．

プログラムは通常のCのアプリケーション・コードと同様，`main()`関数から実行されます．この関数は`src/main.c`に定義されており，初期化の後，ユーザによるコマンド(またはエラー)で終了が指定されるまでループし続けます．ループ内では，ユーザからのコマンドの受け付け，画面の更新を行い，USBイベントのメンテナンスを行います．これらのループ内処理が，先に述べたメンテナンス・ルーチンとして実装されています．

▶ 割り込み処理

WASABIには，もう一つのプログラム・エントリ・ポイントがあります．割り込みルーチンがこれで，`src/_hc_core/isr.c`の中の`isr_isr_USB_Hc()`として定義されています．このルーチンはアプリケーションの初期化時に`hwacces_install_isr()`関数(`src/_hc_hw/Hw_acces.c`内で定義)でインストールされます．

WASABIでは時間の管理を行うだけのために，ISP1362からUSBフレームごとに発生する割り込みを利用しています．このような頻繁な割り込みは本来不要です．CPU負荷の最適化の際には，これを削除できます．

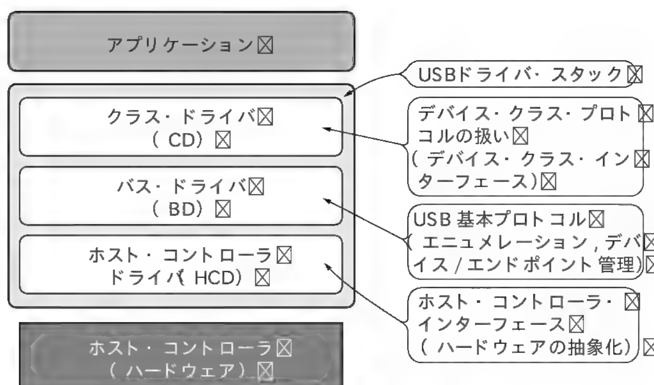


図10　一般的なホスト・スタックの構造

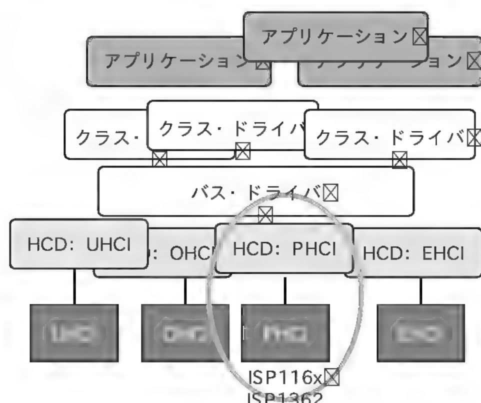


図11　HCDによる各種ホスト・コントローラへの対応

● 一般的なホスト・ドライバ・スタックとの構造の比較

ここから先は，WASABIのさらに細かい部分を見ていくため，まず一般的なホスト・ドライバ・スタックの構造を説明し，これとWASABIとを比較して，各機能がどのように割り当てられているかを見ていきます．

▶ 一般的なホスト・ドライバ・スタックの例

図10に一般的(PC用OSなど)なホスト・ドライバ・スタックの例を示します．

このようなシステムでは下位から上位に向かって，ホスト・コントローラ(Host Controller：HC)，ホスト・コントローラ・ドライバ(Host Controller Driver：HCD)，バス・ドライバ(Bus Driver：BD)，クラス・ドライバ(Class Driver：CD)，アプリケーションというように層をなす構造を持っています．

この構造の中のHCD，BD，CDの部分が，一般にUSBホスト・ドライバ・スタックと呼ばれる部分です．

図中，もっとも下に位置するのはハードウェアで，OHCI/UHCI/EHCIといったものから，ISP1362/ISP116x，ISP176x/ISP1261のようなUSBホスト・コントローラなどに相当します．これらのハードウェアの差異を吸収する抽象化層としてホスト・コントローラ・ドライバ(HCD)が存在します(図11)．

HCDはそれぞれのホスト・コントローラ・アーキテクチャにあわせた具体的なドライバの実装です．このHCD内部でコントローラのレジスタのアクセスや，転送のためのバッファの管理が実装されます．

図12に，組み込みシステム向けホストと，汎用ホストのHCDの違いを示します．組み込み向けホストに内蔵されるバッファ管理などは，すべてこのHCDが処理します．

HCDの上位にはバス・ドライバ(BD)と呼ばれる層が用意されます．この層ではUSBに共通のプロトコルが実装されます．たとえばエニュメレーションの実行や，接続されたデバイス/

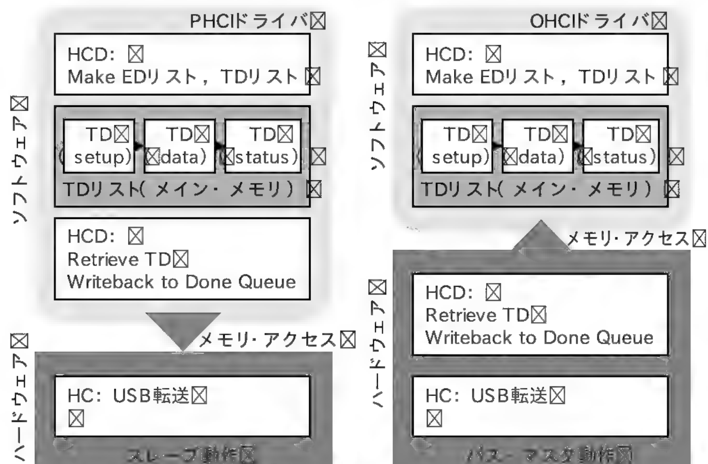


図12　HCDの処理の違い

エンドポイントの管理がこの層で実現されます．

さらにBDの上位にクラス・ドライバ(CD)が用意されます．ここではデバイス・クラスごとに定義されたプロトコルを実装します．デバイス・クラスごとに定義されるエンドポイントの指定や転送手順などがその例です．

図の最上位はアプリケーション層となります．実際のシステムではさらに細かい上位層群としててファイル・システムを扱うファイルI/Oや，より上位のドライバによる前段/後段の信号処理段などが存在します．

▶WASABIの例

WASABIは特定のホスト・コントローラ向けのプログラムであるため，このような細かい層構造は採用していません．図13にWASABIの例を示します．

HCユニット内部はおもに三つの部分に分かれています．HCコアはHCを制御するための主要なサブユニットになっており，一般的なスタック例でいうところのHCDとBDをこの中に含みます．このサブユニット(src/_hc_core/)は表6のような役割を持つモジュールから構成されてます．

この上位にCDが置かれますが，HCコアとの仲立ちをするためのクラス・ドライバ・マネージャ(ClassDr_Mgr)が設けてあります．コード内ではsrc/_hc_clsとして定義されており，クラス・ドライバのインンストールと，クラス依存部分をエニュメレーション処理から抽象化する役目を担っています．

各クラス・ドライバはWASABI独自の実装として，先に述べたようなクラスの種類別に存在します．

● DC_unit/OTG_unitについて

ここではホスト・コントローラの説明を目的としているため，DC_unit，OTG_unitの内部についての詳しい説明は行いません．しかし，図3と図13を見比べると，WASABI内部にはそれぞれのユニットがISP1362のチップ内部の構造にならって用意されているのがわかります．

OTG_unitはOTGプロトコルを実現するコアです．これはOTGのステータスを監視/制御します．このソフトウェアは，ISP1362 DOS評価キットに同梱されるOTG動作検証用アプリケーションOTGCのコードを流用しています．

OTGが動作する際には，ホスト/デバイス両方のソフトウェアが同時に動きます．OTGがホスト側として動作しているときには，ホストはルート・ハブのポート1に接続されたデバイスの制御を行います．このときデバイスは非接続状態に置かれ，サスペンド状態となります．

OTGがデバイス側として動作しているときには，ルート・ハブのポート1が非接続状態になり，デバイス側が通信を行う状態となります．

ポート2はOTGの状態に関わらず，つねにホストに接続された状態を保っているため，継続的に制御することができます．図9はWASABIの典型的なデモの例として紹介しましたが，これはこのような特徴を生かしたセットアップ形態でした．

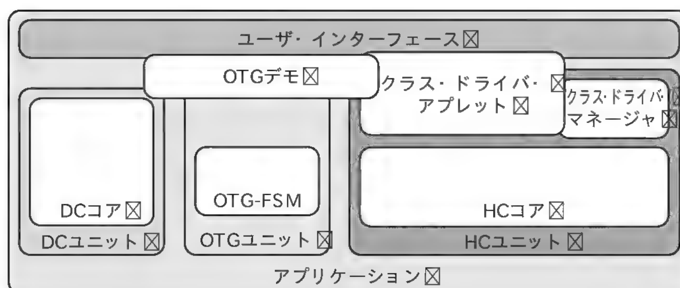


図13　WASABIのモジュール構成

表6　WASABIのモジュールとファイル構成

atl_mix.c	ISP1362内部のATLバッファの管理 (一般的なホスト・スタックのHCDに相当)
dev_ep.c port_ctl.c	デバイス/エンドポイントの管理，ならびにポート・イベントのハンドラ(一般的なホスト・スタックのBDに相当)
transfer.c	コントロール転送の管理
init.c isr.c	HCの初期化処理と割り込みハンドラ

DC制御はフル・スピード・デバイス・コントローラISP1181制御用のコードをそのまま流用できます．WASABIではこのアプリケーション用にISP1181制御コードを新たに作成し，後からホスト，OTGのコードと結合しました．このDCの動作の確認は，もう1台のPCに用意されたWASABI，あるいはISP1181の評価用ドライバを組み込んだPC，あるいは同じチップのホスト・ポートに接続して行うことができます．

● 他プラットホームへの移植

WASABIは，すでにいくつかのシステムに移植された実績があります．ここでは，移植を行う際の基本的な注意点を挙げておきます．

先に述べたようにWASABIは高い移植性を提供するため，プラットホームに依存したコードを，できるだけ分離して記述してあります．また，標準ライブラリの使用などもできるだけ最小限にしてあります．src/_hc_hw/以下にまとめられているコードの変更により，レジスタ・アクセスやハードウェア割り込みのような，他プラットホームへの対応が可能です．

標準ライブラリを使用するようなコードは，src/ui.cにまとめられています．uiモジュールはユーザ・インターフェース関連のコードをまとめたもので，WASABIをデモ・アプリケーションとして動作させるために必要なものです．WASABIをライブラリとして使用する場合には必要ありません．

● 冗長性の除去

WASABIは実験/デモの目的で作製されたコードであるため，コードの記述方法や実行効率，メモリ使用の点などで大きな冗長性をもっています．これらの最適化によって，システム資源をより有効に使うことが可能です．

コードの記述方法については，リアルタイムOSなどを用いる場合，メンテナンス・ルーチンをそれぞれタスクやスレッド

などの実行単位に置き換え，各所に置かれた時間待ち処理をタスクのスリープなどに置き換えることにより，スタック・サイズを節約したり，転送処理中の待ち時間などをより有効に使うことができるでしょう．

また，ヒープ・サイズも削減可能です．WASABIではデバイスを接続した際に行うエニュメレーション処理時に，デバイスのインスタンスを作成します．このとき，デバイスの持つすべてのディスクリプタがこのインスタンス中に保存されます．しかし，これは実験目的で設けられたしくみであるため，削除が可能です．さらに各クラス・ドライバ中で使用されるバッファ・メモリの量の見直しなどがメモリ使用量の最適化の際には有効でしょう．

以上はWASABI自体のコードの構造とメモリ使用の最適化についてでしたが，このほかISP1362を使ううえでのCPUの負荷も考慮の対象となるでしょう．先にも説明したとおり，割り込みの扱いがこの代表的な例です．

WASABIでは時間を管理するだけの目的で，1ms間隔のUSBフレームの開始タイミングで発生する割り込みStart Of Frame(SOF)割り込みを使用しています．本来，ISP1362を使用する上では，このようなSOFによる頻繁な割り込みは不要です．実際の組み込みシステムではシステムが持っているタイマによって時間の管理を行うほうが良いでしょう．システムに用意されたタイマを用いる場合には，SOFによる割り込みを禁止し，転送完了や，ポートのイベントを通知するための割り込みだけを使用するようにします．

ISP1362は転送をトランスファ単位で管理できるため，CPUに対する負荷が非常に小さいという特徴をもっています．トランスファとはパケットやトランザクションより上位の転送単位で，ISP1362では一つのエンドポイントに対して通常1023バイトまでの転送を自動で行うことができます．転送完了はそれぞれのエンドポイントに対する転送完了が割り込みによって通知されます．

CPUに対する負荷をより小さくするため，割り込みをひとつひとつのエンドポイントの転送終了ではなく，複数のエンドポイントの転送終了によって発生させるように設定することが可能です．

最後に，移植を行うにあたって，CPU/システムが扱うデータのエンディアンも注意を要する点です．WASABIは特定のCPU/OSを対象としたアプリケーションであり，リトル・エンディアン・システムで用いられることを前提としています．

このため，リトル・エンディアンのシステムに移植する際には，コードをそのまま用いても問題は発生しません．しかしターゲットのシステムがビッグ・エンディアンの場合，何点かの変更が必要になります(これは残念ながらプラットホーム依存部分として抽象化されていない)．

具体的には`src/_hc_core/dev_ep.c`と，`src/_hc_core/transfer.c`内のUSBディスクリプタを直接操作する部分，ならびに`src/class_dr/storage/`内の各ファイル・システム・データの扱い，ストレージ・コマンドの変換部分にビッグ・エンディアン対応のための変更が必要となります．

5 今後の展開

Philips社では，今後もUSBホスト・コントローラ・ラインナップの拡充を続けていきます．組み込み向けには，先に紹介したISP1362をはじめISP1160/1161を，汎用PCIバス用途向けにはEHCIコアを搭載したISP1561を量産中です．

現在，さらに機能を拡充した次の二つのチップ・ファミリのサンプル供給が開始されています．

● ハイ・スピード対応OTG/ホスト ISP176/ISP1760

ハイ・スピードに対応したISP176xシリーズをサンプル供給中です(2004年7月中旬現在)．ISP1761はISP1362と同様のワンチップOTGコントローラですが，ホスト/デバイスともハイ・スピードに対応した新製品です．ISP1760はISP1761のホスト部分のみのチップとなります．ISP176xのホスト・コントローラはISP1362と同じく，内部に転送用バッファを搭載していますが，より少ないCPU負荷でハイ・スピード・データ転送を実現するために64Kバイトもの大きなRAMを搭載しています．

ISP1362やISP116xではホスト・コントローラ・コアにOHCIを用いていましたが，ISP176xでは，これに代わってハイ・スピードに対応するEHCIが搭載されています．このEHCIコアは，ISP1561ですでに実績のあるPhilips社製EHCIコアにさらに改良を加えたもので，ハイ・スピードで要求されるUSB転送速度の確保と，低システム負荷を同時に実現しています．

デバイス側には，これも定評のあるISP1582のコアを内蔵し，柔軟性と安定した高速転送を実現しています．

このISP176xにも開発キットが用意されています．

● OTGブリッジ ISP1261

ISP1261はOTGブリッジと呼ばれるチップです．このチップはUSBデバイス機器どうしを接続するために生まれました．

図14にその使用例を示します．このようにターゲット・システムがすでにUSB(デバイス機能)を持っていれば，ISP1261をそのUSBポートに接続し，ISP1261のドライバを機器にインストールするだけ[注6]で，USB OTG(またはホスト)機能を実現できます．ISP1261のパッケージは非常に小さいので，基板の空きスペースを利用したり，あるいは従来のUSBコネクタに取り付けるドングル・ケーブルとして実装することが可能です．

図15はISP1362とISP1261を用いた場合の接続の方法の違

注6：ISP1261にはISP1362と同じコアが使われている．このためISP1362のソフトウェアは，ハードウェアの抽象化層だけの変更でISP1261に対応できる．

COLUMN

OTGデュアル・ロール・デバイスと OTGペリフェラル・デバイス

一般的にOn-The-Go対応デバイスというと，ホストにもターゲットにもなれるUSB機器という認識をされていますが，それだけではありません．ホストになれなくても，OTG対応デバイスといえる場合があります．

OTGの仕様では次の2種類のデバイスが定義されています．

1) デュアル・ロール・デバイス
2) ペリフェラル(オンリ)デバイス

OTGデュアル・ロール・デバイスとは，ホストとデバイスの両方の機能を持ち，ホスト・ネゴシエーション・プロトコル(HNP)とセッション・リクエスト・プロトコル(SRP)を実装しているデバイスを指します．一般的に理解されているOTG対応デバイスはこちらになります．

一方，OTGペリフェラル・デバイスとは，SRPのみをサポートしているファンクション・デバイスのことを指しています．OTGのホストがV_{BUS}を切っている状態でターゲット側から通信を行いたい場合に，SRPによりホスト側とのセッションを確立する機能が実装されています．第1章で紹介しているISP1582は，OTGレジスタを介してSRPを行うことが可能になっているので，OTGペリフェラル対応デバイスといえます．

OTGペリフェラル・デバイスがホストになれないのなら，通常のUSBターゲット・デバイスと何が違うのでしょうか．OTG仕様は，何もPDAなどの携帯機器にホスト機能を持たせることだけをねらったものではなく，低消費電力を重視する携帯機器での運用も想定しています．

プロローグなどでも説明したように，USBはターゲット側からホストに対して割り込みをかけるような動作はできません．そこで通常のUSBでは，ターゲットのステータスの変化をインタラプト転送などを使ってホストが定期的にチェックして状態変化を検出します．

しかし，V_{BUS}を切ってホストが完全にスリープ状態に入った場合はどうでしょうか．ターゲットとして接続されている機器側から，ホストに対して何かアクションを起こしたくても，完全にお手上げ状態となります．OTGペリフェラル・デバイスとは，このような状態からでもホストを起こすことのできるデバイスであるといえます．

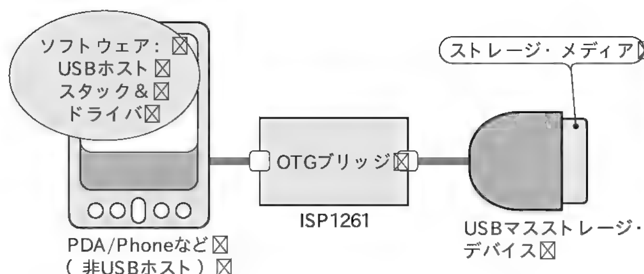
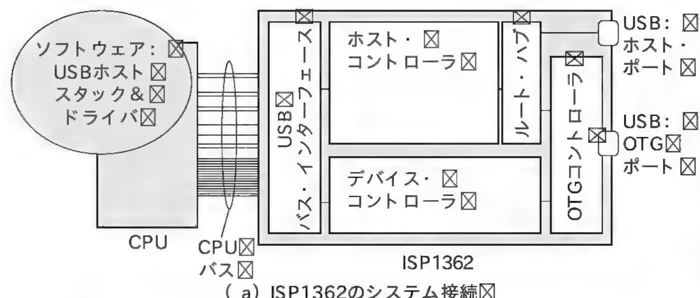


図14　OTGブリッジISP1261の使用例



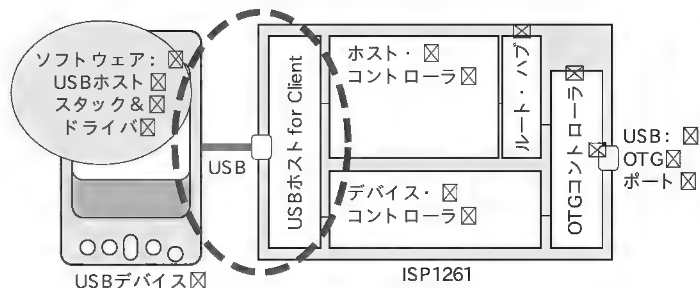


図15　ISP1261を用いたOTG(ホスト)機能の実現

いを示しています．ISP1261はホストとなるシステムへの接続がUSBで行われるため，USBデバイス機器どうしをつなぐブリッジとして働くように見えます．

● リファレンス

最後に，Philips社のUSBデバイスに関連する情報のURLを紹介します．

- Philips Semiconductors社のUSBのページ
  `http://www.semiconductors.philips.com/buses/usb/`
- ISP1362データシート
  `http://www.semiconductors.philips.com/acrobat/datasheets/ISP1362-03.pdf`
- WASABI-Hot!ユーザ・マニュアル
  `http://www.semiconductors.philips.com/acrobat/usermanuals/UM10017-01.pdf`
- USB On-The-Go
  `http://www.semiconductors.philips.com/buses/usb/products/otg/overview/`

おかの・あきふみ　日本フィリップス(株)

P 1 2 3 4 5 6 A1 A2

Appendix

1

USBロゴ認証で必要となるテスト・ツール

USB CVとUSBアナライザを使ったデバッグ技法

谷本 和俊

はじめに

ここでは，Compliance Test(USBロゴ認証)で使用されるテスト・ツール(ソフトウェア)USB CV(USB Command Verifier)を使用したデバッグ方法を紹介します．USB CVは，USB Implementers Forum(USB IF)のWebサイトで公開されており，だれでも入手可能です．USB CVには，実際にCompliance Testの際に使用される「Compliance Test」モードと，選択した一つのテスト項目のみを実行する「Debug」モードがあります(図1)．そのため，USB CVをロゴ認証の際もしくは認証直前になって使用するのではなく，デバッグの初期段階から使用することで問題の切り分けを容易にし，効率の良いデバッグが可能になります．

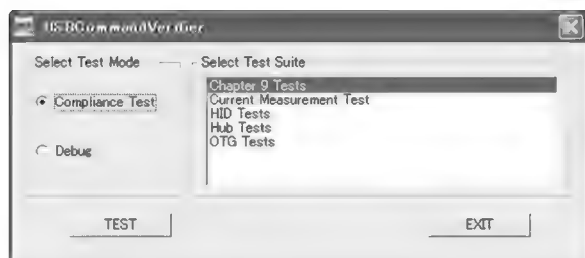


図1　USB CVメイン画面

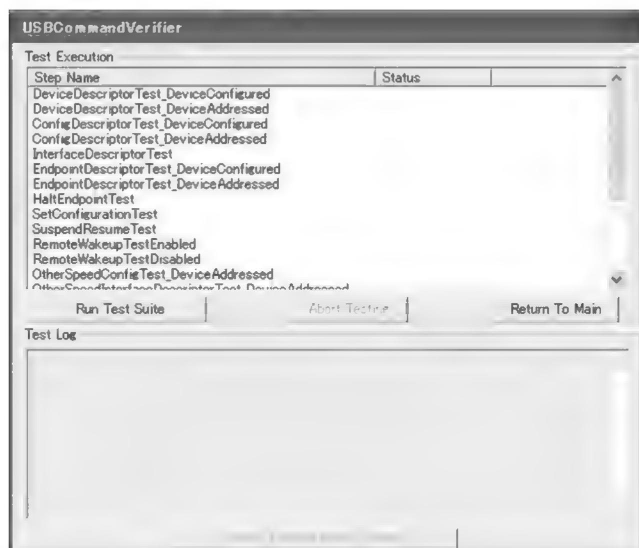


図2　Chapter9 Menu画面

1 デバッグの範囲

USB CVはおもにUSB規格第9章(Chapter 9)で規定されるDevice Frame Workと主要なデバイス・クラスのデータ転送プロトコルのテストが可能です．本章ではすべてのUSBデバイスが守らなくてはならないChapter 9 Tests(図2)を取り上げます．なお，Chapter 9 Testsでは，Endpoint 0のコントロール転送のみが使用され，標準リクエストへの応答が確認できます．

USB CVを使ったデバッグは，図3に示すようにデバッグの初期段階で有効です．デバッグの初期段階でDevice Frame Workの部分をしっかり確認しておくことで，その後のアプリケーション部分のデバッグに専念することができます．

また，いきなりPCなどのUSBホストに接続してデバッグを開始すると，問題があった場合の解析に非常に時間がかかります．この点からも，デバッグの初期段階で，決められたパケットのみを出力するUSB CVの使用が効果的といえます．

2 デバッグの準備

ここではUSB CVを使用するにあたって，事前に準備しなければならない作業を説明します．

▶USB CV実行用PCの用意

USB CVを実行するには，EHCIに準拠したUSB2.0ホスト・コントローラを搭載したPCが必要です．OSは，Windows 2000，Windows XPがサポートされています．Webサイト上でOSは英語版のみと注記されていますが，筆者はWindows XPの日本語環境で動作を確認しました(日本語版Windows 2000では動作確認できていない)．

なお，FS(フル・スピード，12Mbps)，LS(ロー・スピード，1.5Mbps)のデバイスをテスト対象とする場合は，USB2.0準拠ハイ・スピード対応のハブ経由で対象デバイスを接続します．

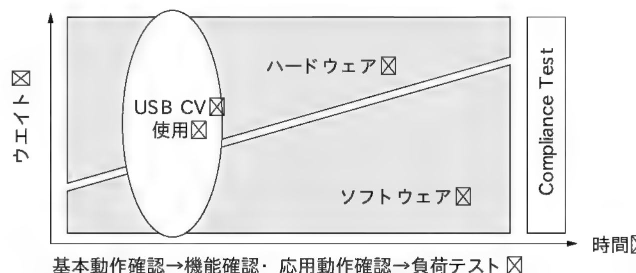


図3　デバッグ範囲



図4　デバイス選択画面



図6　正常起動時のDeviceManager画面

Transaction	S	PID	ADDR	ENDP	DATA
Reset					
GET_DESCRIPTOR [Device]			00	0	
SETUP [DATA0] [ACK]			00	0	8
SETUP	HS	SETUP	00	0	
DATA0	HS	DATA0	00	0	8
0000: 80 06 00 01 00 00 12 00					
ACK	HS	ACK			
IN [DATA1] [ACK]			00	0	18
[illegible] [DATA1] [NYET]			00	0	
SET [illegible]			00	0	
[illegible]			00	0	8
[illegible]			00	0	
GET [illegible]			01	0	
SETUP [DATA0] [ACK]			01	0	8
SETUP	HS	SETUP	01	0	
DATA0	HS	DATA0	01	0	8
0000: 80 06 00 01 00 00 12 00					
ACK	HS	ACK			
IN [DATA1] [ACK]			01	0	18
[illegible]			01	0	
GET [illegible]			01	0	
SETUP [DATA0] [ACK]			01	0	8
SETUP	HS	SETUP	01	0	
DATA0	HS	DATA0	01	0	8
0000: 80 06 00 02 00 00 09 00					
ACK	HS	ACK			
IN [DATA1] [ACK]			01	0	9
PING			01	0	
OUT [DATA1] [NYET]			01	0	
SET_CONFIGURATION			01	0	
SETUP [DATA0] [ACK]			01	0	8
SETUP	HS	SETUP	01	0	
DATA0	HS	DATA0	01	0	8
0000: 00 09 01 00 00 00 00 00					
ACK	HS	ACK			
IN [DATA1] [ACK]			01	0	
GET [illegible]			01	0	
SETUP [DATA0] [ACK]			01	0	8
SETUP	HS	SETUP	01	0	
DATA0	HS	DATA0	01	0	8
0000: 80 06 00 02 00 00 09 00					
ACK	HS	ACK			
IN [DATA1] [ACK]			01	0	9
[illegible]			01	0	
[illegible]			01	0	
GET [illegible]			01	0	
SETUP [DATA0] [ACK]			01	0	8
SETUP	HS	SETUP	01	0	
DATA0	HS	DATA0	01	0	8
0000: 80 06 00 02 00 00 27 00					
ACK	HS	ACK			
IN [DATA1] [ACK]			01	0	39
[illegible]			01	0	
OUT [illegible]			01	0	

図5　起動時のEnumerationシーケンス

▶USB CVの入手とインストール

USB CVは，USB-IFのWebサイト(http://www.usb.org/developers/tools/)からダウンロードします．拡張子.msiのインストーラ形式で公開されているので，ダウンロード後，実行するだけでインストールは完了します．

▶usb.ifファイルの入手とインストール

usb.ifというベンダIDを取得した企業のリスト・ファイルをダウンロードします．URLはUSB CVと同じです．Company Listという項目からダウンロードできます．ダウンロードしたファイルは，USB CVをインストールしたディレクトリにコピーしておきます．このファイルは，Device Descriptorのチェックで使用されます．

以上で準備は完了ですが，実際にテスト項目を実行するためには最低限のEnumerationができる必要があります．テスト実行開始時にVendor ID，Product IDを認識し，テスト対象機器を選択するための処理が行われます．複数のUSBデバイスが接続されていた場合，図4に示すようなデバイス選択画面が現れ，対象機器を選択することができます．図5にテスト実行開始時にUSB上に送出されるパケットのシーケンスを示します．図にあるように，"Get Descriptor"によるDevice Descriptor，Configuration Descriptorの返送と"Set Address"，"Set Configuration"リクエストの受信ができればOKです．

また，USB CV実行時は，通常のEHCI用ホスト・コントローラ・ドライバ(usbehci.sys)からテスト用のホスト・コントローラ・ドライバ(図6)に切り替えが行われます．USB CVが正常に起動できない，テストを実行してもUSBバス上にパケットが出ないままFailで終了するなどの場合は，ドライバの切り替えが正しく行われているかどうかを確認してください．

3 テスト項目と内容

表1にUSB CVのChapter 9 Testsで実行されるテスト項目(Test Step)の一覧とその概要を示します．

1～7項，13～20項は，"Get Descriptor"リクエストによる各種Descriptorの取得および内容の確認が行われます．Step Nameの末尾にテストを実行("Get Descriptor"を発行)するステートが付記されています．それぞれのDescriptor取得をDeviceConfiguredステート，Addressステートで実行します．二つのステートの違いを簡単にいうと，"Set Configuration"リクエストの発行前か後かという違いになります．ステートについての詳細は，USB2.0規格の9.1項にあります．なお，13～20項は，ハイ・スピードをサポートしたデバイスのみに実施されます．

8項は"Set Feature"リクエストによりEndpointをHalt状態にし，"Clear Feature"リクエストによりHalt状態を解除します．それぞれの状態において，"Get Status"リクエストによるEndpoint Statusの確認が行われます．

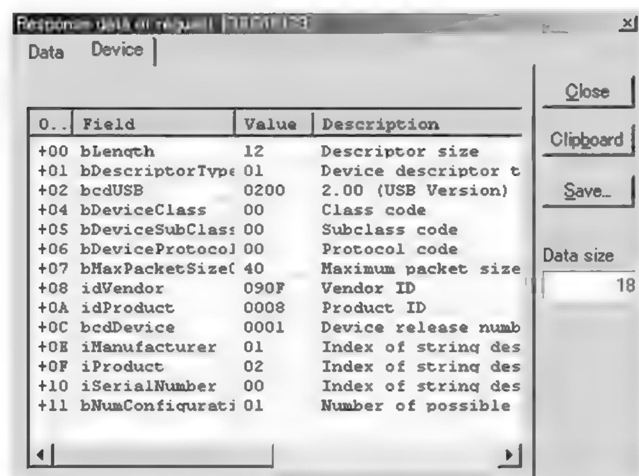
表1　テスト項目一覧

項	Step Name(テスト項目)	概　要
1	DeviceDescriptorTest_DeviceConfigured	Device Descriptorの取得と内容の確認
2	DeviceDescriptorTest_DeviceAddressed	同上
3	ConfigDescriptorTest_DeviceConfigured	Configuration Descriptorの取得と内容の確認
4	ConfigDescriptorTest_DeviceAddressed	同上
5	InterfaceDescriptorTest	Interface Descriptorの取得と内容の確認
6	EndpointDescriptorTest_DeviceConfigured	Endpoint Descriptorの取得と内容の確認
7	EndpointDescriptorTest_DeviceAddressed	同上
8	HaltEndpointTest	Endpoint Halt機能設定/解除の確認
9	SetConfigurationTest	Set ConfigurationによるConfiguration Valueの確認
10	SuspendResumeTest	Suspend/Resume機能の確認
11	RemoteWakeupTestEnabled	Remote Wakeup機能の確認
12	RemoteWakeupTestDisabled	同上
13	OtherSpeedConfigTest_DeviceAddressed	OtherSpeed Configuration Descriptorの取得と内容の確認
14	OtherSpeedInterfaceDescriptorTest_DeviceAddressed	OtherSpeed Interface Descriptorの取得と内容の確認
15	OtherSpeedEndpointDescriptorTest_DeviceAddressed	OtherSpeed Endpoint Descriptorの取得と内容の確認
16	DeviceQualifierTest_DeviceAddressed	Device Qualifier Descriptorの取得と内容の確認
17	DeviceQualifierTest_DeviceConfigured	同上
18	OtherSpeedConfigTest_DeviceConfigured	OtherSpeed Configuration Descriptorの取得と内容の確認
19	OtherSpeedInterfaceDescriptorTest_DeviceConfigured	OtherSpeed Interface Descriptorの取得と内容の確認
20	OtherSpeedEndpointDescriptorTest_DeviceConfigured	OtherSpeed Endpoint Descriptorの取得と内容の確認
21	EnumerateTest	Enumerationシーケンスのループ・テスト

9項は“Set Configuration”リクエストにより指定されたConfiguuration Valueが，“Get Configuration”リクエストで正しく取得できるか確認されます．

10項については，いったんUSBバスをSuspend状態とし，復帰(Resume)後，通常状態(規格書ではnormal operationと記載)に戻ることができるかどうかを確認します．通常状態への復帰は，“Get Descriptor”リクエストに応答できればPassとなります．

11～12項は，Remote Wakeup機能をサポートするデバイス(Configuration DescriptorのbmAttribute．bit5=1のデバイス)のみに実施されます．“Set Feature”，“Clear Feature”リクエストによりRemote Wakeupを有効/無効にした状態で，有効状態のとき”Get Status”リクエストに正しく応答するか，無効状態のときにRemote Wakeupイベントを発生しないかが確認されます．




```
O.. Field              Value  Description
+00 bLength            12     Descriptor size
+01 bDescriptorType    01     Device descriptor t
+02 bcdUSB             0200   2.00 (USB Version)
+04 bDeviceClass       00     Class code
+05 bDeviceSubClass    00     Subclass code
+06 bDeviceProtocol    00     Protocol code
+07 bMaxPacketSize0    40     Maximum packet size
+08 idVendor           090F   Vendor ID
+0A idProduct          0008   Product ID
+0C bcdDevice          0001   Device release numb
+0E iManufacturer      01     Index of string des
+0F iProduct           02     Index of string des
+10 iSerialNumber      00     Index of string des
+11 bNumConfigurati    01     Number of possible
```

図7　Descriptorデコード画面例

21項は2章で紹介した最低限のEnumerationを繰り返すテストです．“Debug Mode”では，ループ回数を指定することができ，Compliance Testでは，150回のループとなります．標準リクエストなどのコントロール転送を割り込み処理としている場合などは，割り込み処理のストレス試験にも利用できます．ただし，データ転送の密度は非常に低いため，USBデバイスとしてのトータルなストレスとはならない点に注意してください．

4　デバッグ手法

各テスト項目について，デバッグ時に確認すべきポイントを述べます．

1～7項，13～20項は，前述したとおり，すべて”Get Descriptor”への応答です．USB CVを使用する前の準備段階で最低限のEnumerationは可能となっているはずなので，ここでは各Descriptorを返送するハンドラを漏れなく用意してあるか，Descriptorの定義内容にまちがいがないかが確認のポイントとなります．USB 2.0で追加されたDevice Qualifier Descriptor，Other Speed Configuration Descriptorなどに注意が必要です．このDescriptorの内容確認でUSBアナライザを使用すると，デバッグ時間の短縮に効果を発揮します．標準的なUSBアナライザであれば，標準Descriptorのデコード機能があります(図7にデコード画面例を示す)．値の意味やワード・データのエンディアンなどを自動的に解釈してくれるため，ここでの確認に有用となります．

8～12項がUSB CVをデバッグに使用することで効力を発揮する点です．PCを通常動作(動作状態)させているだけでは発生しない状態を確認することが可能となります．近年，ソフトウェア開発の負担を軽減するために，標準リクエストに対して自動的に応答を返す

ことが可能なUSBコントローラICが増えています．それらの機能を効果的に使用することも一手です．しかし，その場合でもデバッグ初期段階での確認は必要です．

8，9項はそれぞれ設定された値と応答を確認する必要がありますが，ここでもUSBアナライザを使用することで簡単に確認することができます．8項では，Set Feature後のGet StatusおよびClear Feature後のGet Statusで応答する値（Endpoint Status）を確認します．9項では，Set Configurationで指定されたConfiguration値をGet Configurationで正しく応答するかを確認します．図8に9項のテスト実行時をキャプチャしたアナライザ画面を示します．

11，12項のテスト実行には，Remote Wakeupイベントを発生させるというユーザ操作が必要となります．したがって，ほかのソフトウェア・モジュールやシステムのハードウェアの動作に影響を受ける部分といえます．この項目だけは，ここまで紹介したようなデバッグの初期段階では実行しづらい項目かもしれません．ただし，Remote Wakeup機能をサポートする機器にとっては，重要な項目なので，ダミーのターゲット（ハードウェア，ソフトウェア含め）を用意するなどして，初期段階で試しておいてほしい項目です．

5 その他のテスト項目

USB CVには本項で紹介したChaptor9 Testのほかに，次のデバイスについてテストが実行可能です．

- HID（Human Interface Device）
- Hub（Chapter 11）
- OTG（USB On-The-Go）

これらのテストはそれぞれの規格で定められたデータ転送プロトコルの確認が行われます．

また，2004年中にMass StorageクラスのCompliance Test開始がアナウンスされています．Mass Storageクラスについては，現在Working GroupメンバにUSB CVに追加するモジュールが公開され議論されています．近々Mass StorageデバイスのCompliance Testも開始されることでしょう．

まとめ

今回ご紹介したUSB CVのほかに，USB IFのWebサイトには任意のアドレス/エンド・ポイントにパケットを出力するSingle Step Transaction Debugger（SSTD）なども公開されています（ただし，筆者の日本語環境では動作が確認できなかった）．これらのツールを効果的に使用することで，相互接続性の高い機器を開発することやターゲットが完成していない段階でのデバッグが可能です．

これらのツールやテスト内容は変更されることもあるので，USB IFのWebサイトは定期的にご覧になることをお勧めします．

本稿が，USB機器のデバッグ効率向上に少しでもお役に立てれば幸いです．

```
09 SetConfigurationTest-cut.utd
S S    PID    Packet name                     A    E   DATA
HS 00  SETUP                                  01   0
HS 00  DATA0  SET_CONFIGURATION               01   0   8
              0000:  00 09 01 00 00 00 00 00
HS 00  ACK
HS 00  IN                                     01   0
HS 00  DATA1                                  01   0   0
HS 00  ACK
HS 00  SETUP                                  01   0
HS 00  DATA0  GET_CONFIGURATION               01   0   8
HS 00  ACK
HS 00  IN                                     01   0
HS 00  DATA1                                  01   0   1
              0000:  01
HS 00  ACK
HS 00  OUT                                    01   0
HS 00  DATA1                                  01   0   0
HS 00
HS 00  SETUP                                  01   0
HS 00  DATA0  SET_CONFIGURATION               01   0   8
              0000:  00 09 00 00 00 00 00 00
HS 00  ACK
HS 00  IN                                     01   0
HS 00  DATA1                                  01   0   0
HS 00  ACK
HS 00  SETUP                                  01   0
HS 00  DATA0  GET_CONFIGURATION               01   0   8
HS 00  ACK
HS 00  IN                                     01   0
HS 00  DATA1                                  01   0   1
              0000:  00
HS 00  ACK
HS 00  OUT                                    01   0
HS 00  DATA1                                  01   0   0
HS 00
HS 00  SETUP                                  01   0
HS 00  DATA0  SET_CONFIGURATION               01   0   8
              0000:  00 09 01 00 00 00 00 00
HS 00  ACK
HS 00  IN                                     01   0
HS 00  DATA1                                  01   0   0
HS 00  ACK
HS 00  SETUP                                  01   0
HS 00  DATA0  GET_CONFIGURATION               01   0   8
HS 00  ACK
HS 00  IN                                     01   0
HS 00  DATA1                                  01   0   1
              0000:  01
HS 00  ACK
HS 00  OUT                                    01   0
HS 00  DATA1                                  01   0   0
HS 00
```

図8　SetConfigurationTestのシーケンス

たにもと・かずとし　富士通デバイス（株）

Appendix 2

ディジタル・カメラとプリンタをダイレクトにつないで印刷できる

PictBridge 規格の概要

佐藤 陽二

1 PictBridge 規格とは

● PictBridgeの背景

近年，デジカメが普及し，画像をディジタル・データとして手軽に扱えるようになってきました．しかし，これまで，デジカメで撮った画像を印刷するには，デジカメをパソコンとつないで，印刷する画像データをいったんパソコンに取り込み，その後，パソコンからプリンタに印刷するという作業が必要でした．すなわち，印刷するには，パソコンが必要ですし，パソコンの立ち上げや，専用の印刷ソフトを操作する手間がかかり，だれでも簡単に印刷できるというわけではありませんでした(図1)．

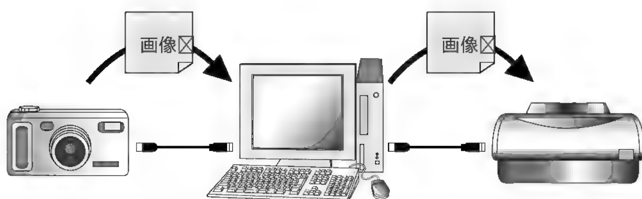


図1 従来方式による画像印刷

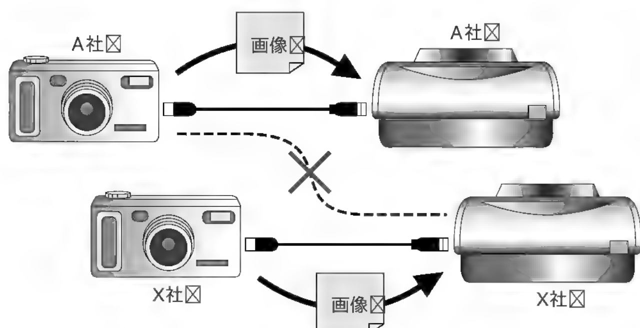


図2 各社独自方式による画像の直接印刷

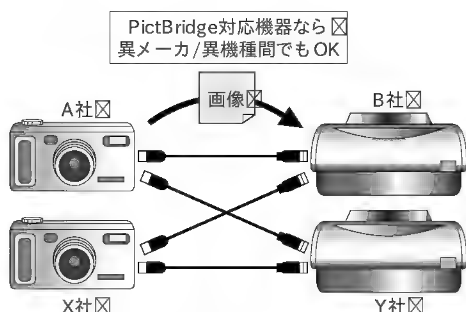


図3 PictBridgeによる異メーカ/機種間の画像印刷

これを解決するために，パソコンを経由せず，デジカメとプリンタを直接つなぎ，そのまま印刷できる機器がいくつかのメーカから製品化されました．ただし，それぞれのメーカが独自のプロトコルを用いてデジカメとプリンタを接続したため，違うプロトコルを用いたメーカの機器では，直接印刷できないという問題がありました(図2)．

この問題を克服すべく，デジカメ，プリンタ・メーカ6社(キヤノン(株)，富士写真フィルム(株)，ヒューレット・パッカードカンパニー，オリンパス光学工業(株)，セイコーエプソン(株)，ソニー(株)：以上，アルファベット順)が集まり，メーカに依存しないオープンな標準規格としてPictBridgeが提案されました．その後，規格の管理，運営業務がCIPAへ委託され，「CIPA規格 CIPA DC-001-2003 Digital Photo Solutions for Imaging Devices」として，2003年2月3日に正式リリースされました．

これにより，PictBridgeを採用した機器間であれば，簡単に画像を印刷することが可能になりました(図3)．

● PictBridgeの機能

PictBridgeでは，印刷対象，印刷方法の指定機能や，印刷状態の監視，印刷実行制御機能などを提供し，印刷したい画像を簡単に，かつ，いろいろな方法で印刷することができます．表1にPictBridgeのおもな機能を示します．

2 PictBridgeのシステム構成

● システム機器構成

PictBridgeでは，図4に示すようにUSBインターフェースで画像入力デバイスと画像出力デバイスを直接接続します．USBインターフェースにおいて，画像入力デバイスはUSBデバイス側，画像出力デバイスはUSBホスト側として動作します．

● 内部アーキテクチャ

PictBridgeの内部アーキテクチャを図5に示します．

表1 PictBridgeの機能

	機 能	内 容
1	印刷対象指定	●対象画像の選択指定 (モニタ上などで選択された画像) ●DPOFファイルによる対象画像指定 ●全画像
2	印刷方法指定	●通常印刷 ●インデックス・プリント ●切り抜き印刷 ●枚数指定印刷 ●日付け付加 ●画像サイズ指定
3	印刷状態監視	●接続確立通知 ●印刷状態/終了/エラー通知
4	印刷実行制御	●印刷中止 ●エラー時の印刷再開

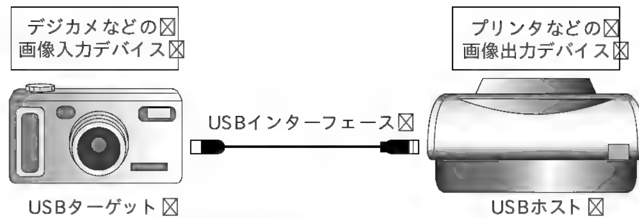


図4　PictBridge 対応機器の接続

表2　SICDのエンドポイント構成

	用　途	エンドポイント種別
1	制御用	コントロール(default)
2	データ転送用	バルク IN
3		バルク OUT
4	イベント通知用	インタラプト IN

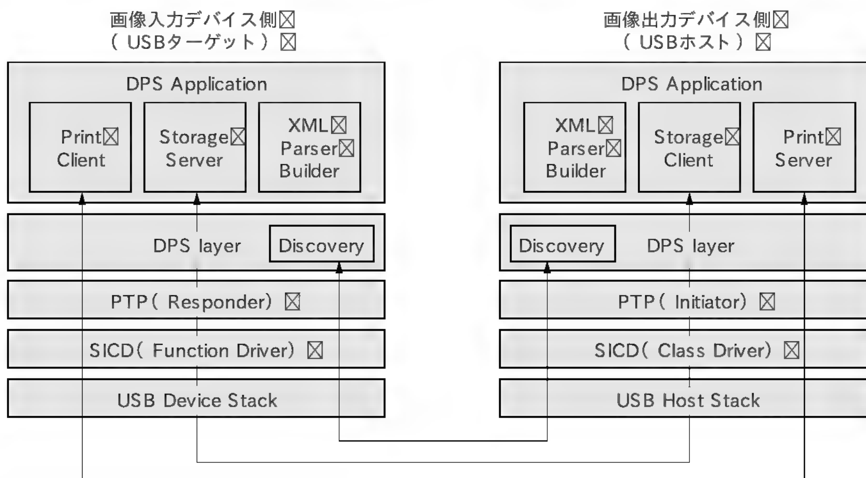


図5　PictBridge 内部アーキテクチャ

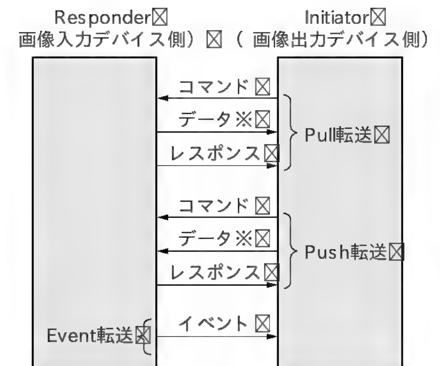


図6　PTP における機器の役割と各種転送

▶ USB Host/Device Stack

USBインターフェースによるデータ転送を行うためのモジュールで，画像入力デバイス側がUSB Device Stack，画像出力デバイス側がUSB Host Stackを搭載します．

▶ SICD(Still Image Capture Device) Class/Function Driver

USBインターフェース上にPTPプロトコルを実装するためのクラス・ドライバ・モジュールです．SICDクラスは表2に示す四つのエンドポイントを使って通信を行います．

▶ PTP Initiator/Responder

PTP(PIMA 15740 2000)規格に準拠した画像転送プロトコル・モジュールで，画像データおよび制御データを該当するタイプのオブジェクトとして受け渡します．

PTPでは要求を発行する側をInitiator，応答する側をResponderと呼び，PictBridgeにおいては画像出力デバイス側がInitiator，画像入力デバイス側がResponderとなります．

PTPにおけるデータ転送の種類には，Initiator側でデータを受け取るPull，Initiator側からデータを送り出すPush，Responder側から短い情報を通知するEventの3種があり，Push，Pull転送の各トランザクションは，コマンド・フェーズ，データ・フェーズ，レスポンス・フェーズで構成されます(コマンドの種別によってはデータ・フェーズがない場合もあり)．Event転送はイベント・フェーズのみです(図6)．

▶ DPS layer

PictBridgeのプロトコル通信をPTPにマッピングするための変換レイヤで，接続時のDPS Discovery機能および上位DPSアプリケーションに対するDPS Operation，DPS Eventの転送機能を提供します．

▶ DPS Application

(a) Print Server/Client

画像入力デバイス側からの印刷要求を処理するためのモジュールで，画像入力デバイス側がPrint Client，画像出力デバイス側がPrint Serverとなります．

(b) Storage Server/Client

画像出力デバイス側からの画像データ取得要求を処理するためのモジュールで，画像入力デバイス側がStorage Server，画像出力デバイス側がStorage Clientとなります．

(c) XML Builder/Parser

上記Server/Client間のDPS Operation，DPS EventはXML形式のファイルのやり取りで実現されます．そのXMLデータを生成，解析するために画像入力デバイス，画像出力デバイスの双方にXML BuilderおよびParserモジュールが必要です．

● PictBridgeのオペレーション/イベント

PictBridgeで定義されているDPS OperationおよびDPS Eventを表3に示します．

3 PictBridgeの動作概要

● PictBridgeの全体動作フロー

図7にPictBridge全体の動作フローを示します．

(1) Discovery

USBケーブルの接続をトリガとして，まずUSBのバス・エニュメレーション処理が動作します．そこでSICDクラス・デバイスとして接続の認識が行われます．次に，PTP layerのセッション確立が行わ

れます．その後，DPS layerのDPS_Discoveryにより，互いの機器がDPS機能を有する機器かどうかのネゴシエーションを行います．双方の機器がPictBridge機器であることを認識した後，DPSアプリケーションに制御が移行します．

表3　Print Service Operation

	オペレーション	内　容
1	DPS_ConfigurePrintService	バージョン情報およびその他コンフィグレーション情報の交換
2	DPS_GetCapability	プリンタ側の能力情報取得
3	DPS_GetJobStatus	ジョブ・ステータスの取得
4	DPS_GetDeviceStatus	デバイス・ステータスの取得
5	DPS_StartJob	コンフィグレーションの確立とプリント・ジョブの開始
6	DPS_AbortJob	全プリント・ジョブの中断
7	DPS_ContinueJob	プリント・ジョブの再開

(a) Print Service Operation

	イベント	内　容
1	DPS_NotifyJobStatus	プリント・ジョブの状態変化通知
2	DPS_NotifyDeviceStatus	プリンタ・デバイスの状態変化通知

(b) Print Service Event

	オペレーション	内　容
1	DPS_GetFileID	ファイルID取得
2	DPS_GetFileInfo	ファイル情報取得
3	DPS_GetFile	ファイル取得
4	DPS_GetPartialFile	ファイル分割取得
5	DPS_GetFileList	ファイル・リスト取得
6	DPS_GetThumb	サムネイル取得

(c) Storage Service Operation

(2) Configure

DPSアプリケーションが起動すると，まず最初に，DPS_ConfigurePrintServiceによって，接続した機器どうしが必要な機能を有していることを確認します．その後，それぞれのServer/Client間で接続が確立され，以降DPS Operation，DPS Eventの通信が行われます．

(3) GetCapability

画像入力デバイス側からDPS_GetCapabilityで画像出力デバイス側に設定可能な能力を問い合わせます．得られた情報はUIなどに適宜反映させ，ユーザが必要に応じて選択できるようにします．以上で画像入力デバイス側はPictBridge対応機器として印刷の制御ができるようになります．

(4) StartJob

ユーザが画像入力デバイス側で印刷の開始操作をすることによりDPS_StartJobが発行され，画像出力デバイス側に各種印刷設定情報とともに印刷開始要求が渡されます．

(5) GetFileInfo，GetFile

画像出力デバイス側は，画像入力デバイス側にDPS_GetFileInfoで印刷に必要な画像ファイルのサイズなどの情報を要求し，その情報に基づいてDPS_GetFileで実際の画像データ・ファイルを読み込み印刷処理を行います．

なお，画像出力デバイス側で画像データ・ファイルを一気に読み込むことができないような場合には，DPS_GetPartialFileにより画像データ・ファイルを分割して読み込むことも可能です．

(6) Notify

画像出力デバイス側は，指定された印刷が終了した後，その結果をDPS_NotifyDeviceStatusを使って画像入力デバイス側に通知します．

最初の印刷は上記(1)～(6)のシーケンスを順次行って終了し，以降の印刷は(4)～(6)のシーケンスを繰り返すことで実行できます．

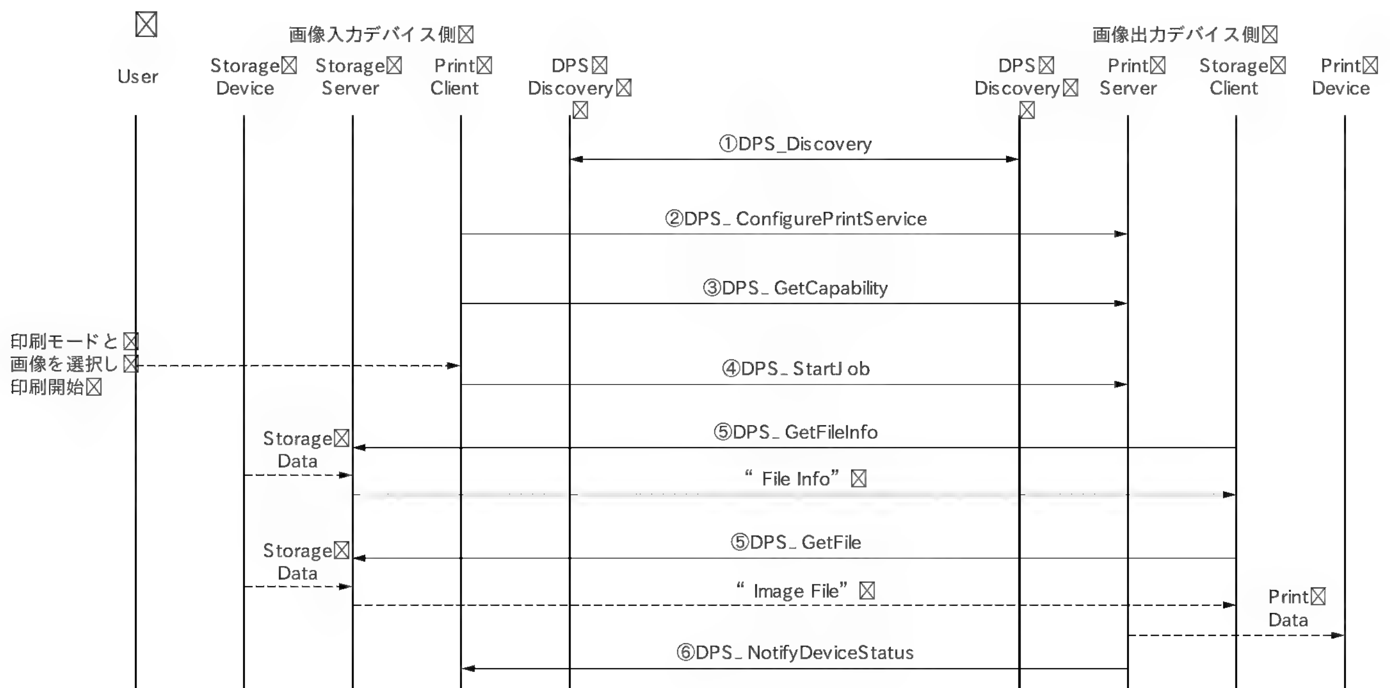


図7　PictBridgeの全体動作フロー

● 内部処理フロー

以下 PictBridge 内部の動作シーケンスをいくつかの代表的なコマンドを例に説明します．

(1) DPS_StartJob（図8）

(a) 画像入力デバイス側でユーザから印刷が要求されると，まず XML Builder を使って DPS_StartJob の XML ファイルを生成します．次に，PTP の RequestObjectTransfer イベントで画像出力デバイス側に XML ファイルが生成されたことを通知します．

(b) RequestObjectTransfer を受け取った画像出力デバイス側は，まず，PTP の GetObjectInfo コマンドでファイルの情報を取得します．次に，取得したファイル情報をもとに GetObject で XML ファイルを受け取ります．その後，XML Parser によりコマンドの解析を行い，該当する処理を起動します．この場合は DPS_StartJob をプリンタの上位アプリケーション側に通知します．

(c) DPS_StartJob を受け付けた画像出力デバイス側は，まず，XML Builder を使って DPS_StartJob 応答用の XML ファイルを作成します．次に，PTP の SendObjectInfo コマンドで画像入力デバイス側にファイル情報を送信します．その後，SendObject コマンドで XML ファイルを送信します．

(d) XML ファイルを受信した画像入力デバイス側は，XML Parser を使ってコマンドの解析を行い，該当する処理を起動します．この場合は，DPS_StartJob の応答による印刷要求の受付完了処理が行われます．

(2) DPS_GetFile（図9）

DPS_StartJob を受け付けた画像出力デバイス側は，まず DPS_GetFileInfo を発行します．次に，PTP の GetObjectInfo により画像ファイルの情報を取得します．さらに，取得したファイル情報をもとに DPS_GetFile を発行し，PTP の GetObject により画像ファイルを取得します．その後，取得した画像ファイルの印刷を実行します．

(3) DPS_NotifyDeviceStatus（図10）

(a) 画像出力デバイス側で，DPS_NotifyDeviceStatus によりデバイス状態の通知を行う場合，まず，XML Builder を使って DPS_NotifyDeviceStatus の XML ファイルを生成します．次に，PTP の SendObjectInfo コマンドで画像入力デバイス側にファイル情報を送信します．その後，SendObject コマンドで XML ファイル

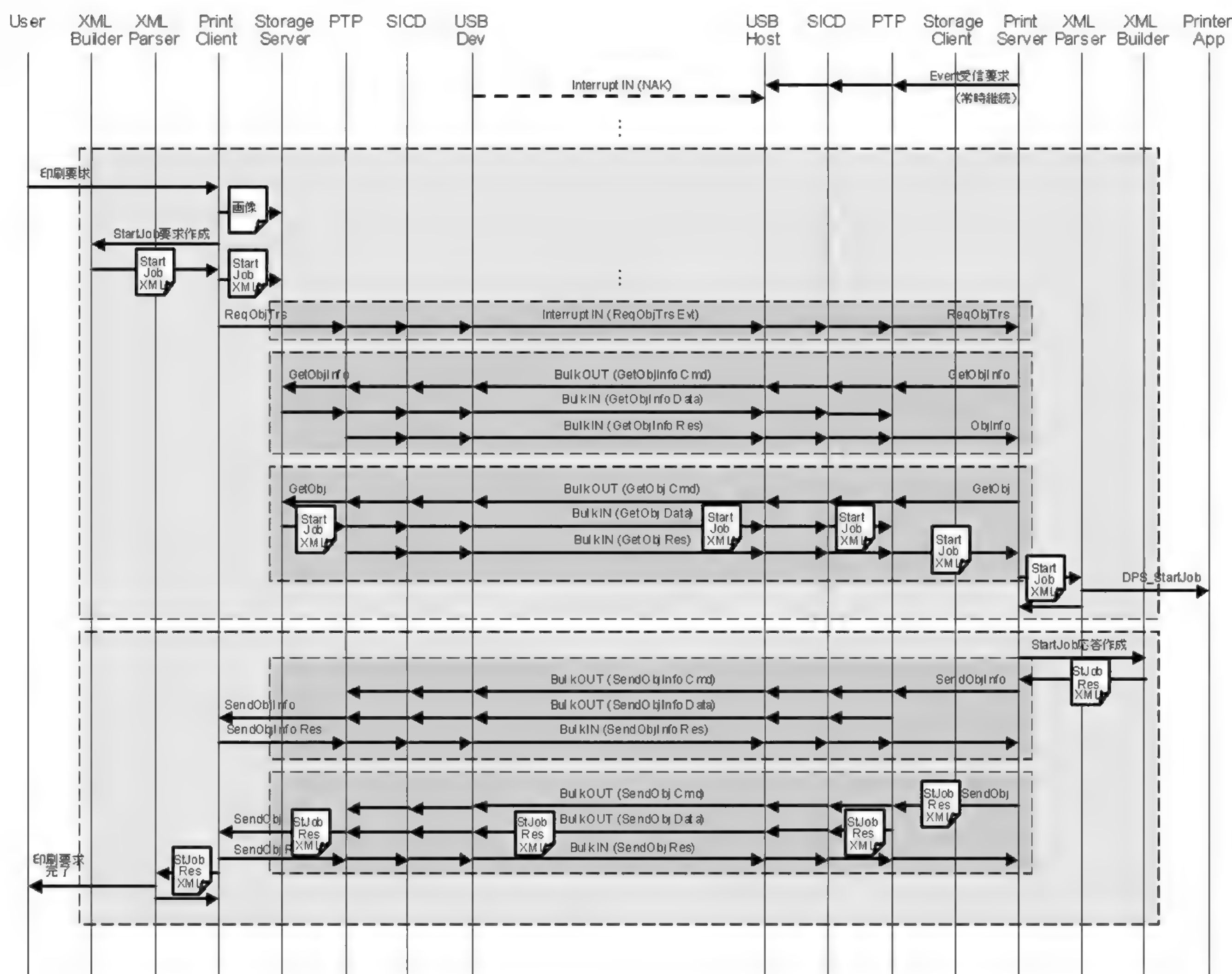


図8　DPS_StartJob の動作シーケンス例

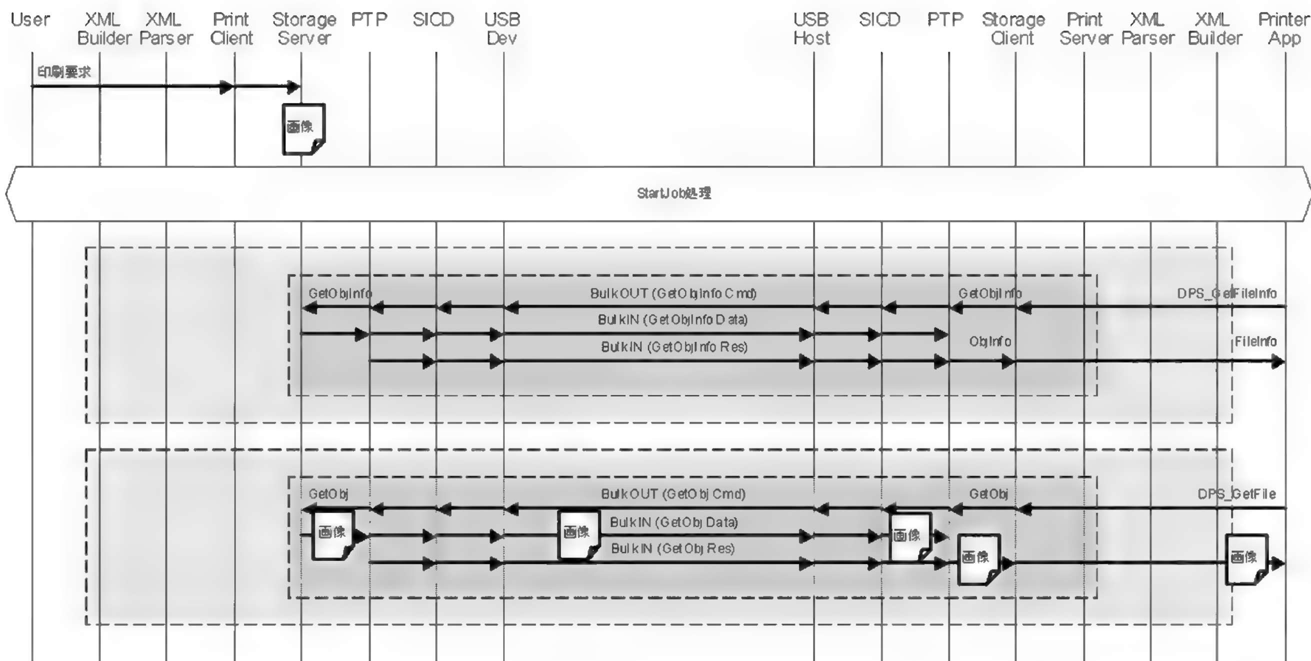


図9　DPS_GetFileの動作シーケンス例

User
XML Builder
XML Parser
Print Client
Storage Server
PTP
SICD
USB Dev
USB Host
SICD
PTP
Storage Client
Print Server
XML Parser
XML Builder
Printer App
DPS_NotifyDeviceStatus
NotifyDevStat要求作成
Notify DvSt XML
BulkOUT (SendObjInfo Cmd)
SendObjInfo
SendObjInfo
BulkOUT (SendObjInfo Data)
SendObjInfo Res
BulkIN (SendObjInfo Res)
BulkOUT (SendObj Cmd)
Notify DvSt XML
SendObj
SendObj
BulkOUT (SendObj Data)
SendObjR
BulkIN (SendObj Res)
Notify DvSt XML
Notify DvSt XML
Notify DvSt XML
Notify DvSt XML
デバイス状態通知
NotifyDevStat応答作成
NDS Res XML
NDS Res XML
ReqObjTrs
InterruptIN (ReqObjTrs Evt)
ReqObjTrs
GetObjInfo
BulkOUT (GetObjInfo Cmd)
GetObjInfo
BulkIN (GetObjInfo Data)
BulkIN (GetObjInfo Res)
ObjInfo
GetObj
BulkOUT (GetObj Cmd)
GetObj
BulkIN (GetObj Data)
BulkIN (GetObj Res)
NDS Res XML
NDS Res XML
NDS Res XML
NDS Res XML
NDS Res XML
DPS_NotifyDeviceStatus 応答

図10　DPS_NotifyDeviceStatusの動作シーケンス例

を送信します．

(b) XMLファイルを受信した画像入力デバイス側は，まず，XML Parserを使ってコマンドの解析を行い，該当する処理を起動します．この場合はDPS_NotifyDeviceStatusによるデバイス状態の通知受付処理が行われます．

(c) DPS_NotifyDeviceStatusを受け付けた画像入力デバイス側は，まず，XML Builderを使ってDPS_NotifyDeviceStatus応答用のXMLファイルを生成します．次に，PTPのRequest ObjectTransferイベントで画像出力デバイス側にXMLファイルを作成したことを通知します．

(d) RequestObjectTransferを受け取った画像出力デバイス側は，まず，PTPのGetObjectInfoコマンドでファイルの情報を取得します．次に，取得したファイル情報をもとにGetObjectでXMLファイルを受け取ります．その後，XML Parserによりコマンドの解析を行い，該当する処理を起動します．この場合はDPS_NotifyDeviceStatus応答受信によりDPS_NotifyDeviceStatusの完了をプリンタの上位アプリケーション側に通知しています．

● PictBridgeのXMLファイル

PictBridgeのオペレーションやイベントはXML形式のファイルとして受け渡されます．代表的なオペレーションのXMLファイルの例をリスト1～リスト4に示します．

● PictBridgeに関する情報

PictBridgeに関する規格書やガイドラインなどの入手方法およびPictBridgeに関するさらなる情報は，次のWebページより入手可能です．

http://www.cipa.jp/pictbridge/

参考文献

▶ PictBridge

(1) White Paper of CIPA DC-001-2003 Digital Photo Solutions for Imaging Devices (Japanese) / February 3, 2003

(2) CIPA DC-001-2003 Digital Photo Solutions for Imaging Devices / February 3, 2004.

(3) Implementer's Guideline For CIPA DC-001-2003 / April 4, 2003.

▶ PTP

(4) PIMA 15740 : 2000 First Edition / July 5, 2000.

▶ SICD

(5) Universal Serial Bus Still Image Capture Device Definition Revision 1.0 / July 11, 2000.

▶ USB2.0

(6) Universal Serial Bus Specification Revision 2.0 / April 27, 2000.

さとう・ようじ　(株)グレープシステム

リスト1　DPS_StartJob要求のXMLファイル例

```
<?xml version="1.0"?>
<dps xmlns="http://www.cipa.jp/dps/schema/">
  <input>
    <startJob>
      <jobConfig>
        <quality>XXXXXXXX</quality>
        <paperSize>XXXXXXXX</paperSize>
        <fileType>XXXXXXXX</fileType>
        <datePrint>XXXXXXXX</datePrint>
        <fileNamePrint>XXXXXXXX</fileNamePrint>
        <imageOptimize>XXXXXXXX</imageOptimize>
        <layout>XXXXXXXX</layout>
        <cropping>XXXXXXXX</cropping>
      </jobConfig>
      <printInfo>
        <croppingArea>XXXX XXXX XXXX XXXX</croppingArea>
        <fileID>XXXXXXXX</fileID>
        <fileName>FILENAME</fileName>
        <date>DD MMM, YYYY</date>
      </printInfo>
    </startJob>
  </input>
</dps>
```

リスト2　DPS_StartJob応答のXMLファイル例

```
<?xml version="1.0"?>
<dps xmlns="http://www.cipa.jp/dps/schema/">
  <output>
    <result>XXXXXXXX</result>
    <startJob/>
  </output>
</dps>
```

リスト3　DPS_NotifyDeviceStatus要求のXMLファイル例

```
<?xml version="1.0"?>
<dps xmlns="http://www.cipa.jp/dps/schema/">
  <input>
    <notifyDeviceStatus>
      <dpsPrintServiceStatus>XXXXXXXX</dpsPrintServiceStatus>
      <jobEndReason>XXXXXXXX</jobEndReason>
      <errorStatus>XXXXXXXX</errorStatus>
      <errorReason>XXXXXXXX</errorReason>
      <disconnectEnable>XXXXXXXX</disconnectEnable>
      <capabilityChanged>XXXXXXXX</capabilityChanged>
      <newJobOK>XXXXXXXX</newJobOK>
    </notifyDeviceStatus>
  </input>
</dps>
```

リスト4　DPS_NotifyDeviceStatus応答のXMLファイル例

```
<?xml version="1.0"?>
<dps xmlns="http://www.cipa.jp/dps/schema/">
  <output>
    <result>XXXXXXXX</result>
    <notifyDeviceStatus/>
  </output>
</dps>
```

COLUMN

PictBridge 機器のデバッグが可能な USB アナライザ

谷本 和俊

●PictBridge 機器のデバッグ

USB を利用した PictBridge 機器のデバッグでは，USB 上のデータ転送だけではなく，SICD(Still Image Capture Device Class)，PTP(Picture Transfer Protocol)，DPS(Digital Photo Solution for Imaging Devices)それぞれのレイヤでの解析が求められます．ここでは USB アナライザ USB ZERONE(富士通デバイス)の PictBridge 機器デバッグに有用な機能を紹介します．

●フラグメント・データの復元

USB を利用した PictBridge のデータ転送では，USB 規格に則って，転送するデータがエンドポイントのパケット・サイズ単位に分割(フラグメント)されます．連続する同種のトランザクションからデータ部分のみを抽出し，まとめて表示します．PictBridge 機器の解析では，SICD(PTP)で規定されるヘッダ・データも認識し，実データのみを表示することが可能です(図 A)．

この機能を応用して，転送された JPEG データを復元することができます．GetPartialObject での JPEG データ転送時は，オフセットと転送サイズを認識して復元するため，表示可能な JPEG ファイルに保存することが可能です．

●XML スクリプトの確認

前述フラグメント・データの復元機能をさらに応用し，XML スクリプトを表示することが可能です．GetObject，SendObject でスクリプト・データが転送されたことを認識し，XML 表示(図 B)が可能となります．この機能と検索機能を組み合わせることにより，XML スクリプトのやり取りだけを順次確認していくことができます．

●PictBridge 固有のパケット検索

検索機能としては，SICD レイヤの検索と DPS レイヤの検索(図 C)が可能です．どちらのレイヤについてもパラメータごとに検索対象とするか否かの指定ができます．

たにもと・かずとし　富士通デバイス(株)

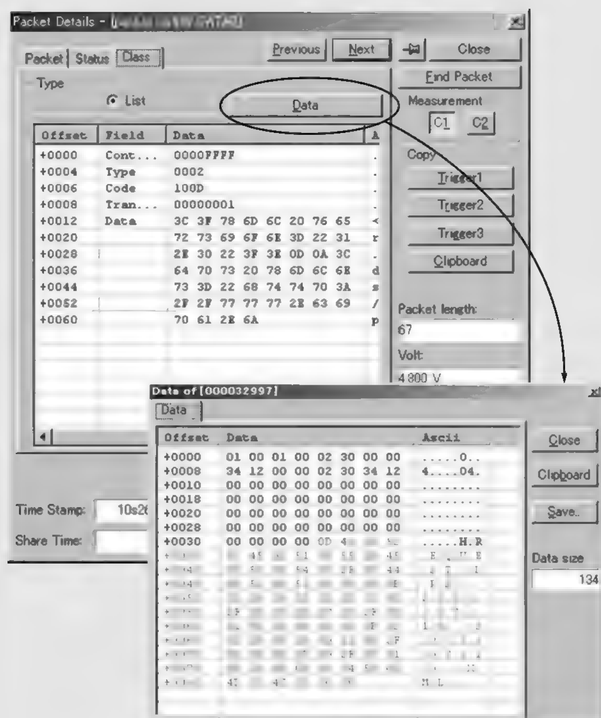


図 A　フラグメント・データ復元画面

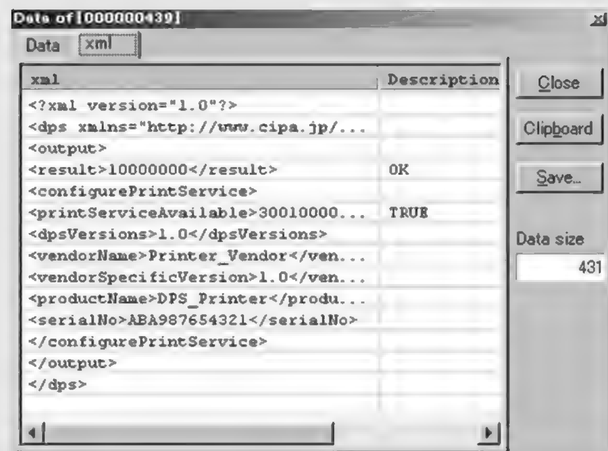


図 B　XML 表示画面

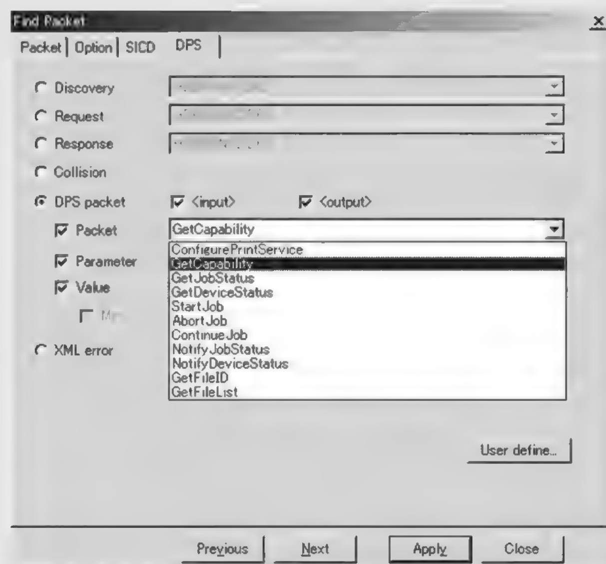


図 C　DPS レイヤ検索画面

ツールを使って開発期間の短縮を図ろう

WinDriverを使ったデバイス・ドライバの開発

西 伸顕

本稿では，Jungo社製のWinDriverというツールを利用して，デバイス・ドライバを開発する手法を紹介します．

一般に，デバイス・ドライバを開発するためには，OSの内部構造[注1]や，ドライバ・インターフェース[注2]などの基礎知識が必要となります．また，開発には多くの手順を踏むうえに，各OSに応じて開発する必要があるので，たくさんの作業と時間が必要になります．

WinDriverを利用すると，多くの手順が自動化できるうえに，上記のようなOSの知識の習得が不必要になるために，開発期間の短縮が図れます．

さらに，WindowsやLinux，Solaris，VxWorksといったOS間で互換性のあるソース・コードを記述することも可能です．

WinDriverの評価版は，エクセルソフト(株)のWebサイトから無償でダウンロードできます．

http://www.xlsoft.com/jp/products/download/

1 WinDriverと開発手順の概要

● WinDriverの基本構造

WinDriverは，カーネル・モードではなく，ユーザ・モードで開発を行います(図1)．また，ウィザードでハードウェアの診断を行い，自動的にハードウェア独自のドライバ・コードを生成できます．ハードウェア独自のドライバ・コード(アプリケーション)は，WinDriverが提供するカーネル・レベル(Ring0)で動作する汎用的なカーネル・モジュールを使用します．このカーネル・モジュールが提供するハードウェアへのアクセスを行うAPIを使用して，アプリケーションはハードウェアのレジストリ，メモリ範囲，I/O範囲へのアクセス，ハードウェアの割り込みの処理(PCI/ISAの場合)，およびデバイスのパイプのデータ転送(USBの場合)を行います．

● 従来までの手間と時間のかかる開発手法

今までの一般的なドライバの開発は，次のような手順で行われてきました．

1) OSの内部構造を学習(Windows，Linux，VxWorksなど)
2) 各OSでのデバイス・ドライバの記述方法を学習(DDKなど)
3) カーネル・モードの開発や，デバッガ・ツールの使用方法を習得
4) カーネル・モードのデバイス・ドライバを記述(基本的なハードウェアのI/O部)
5) ユーザ・モードでアプリケーションを記述(カーネル・モードで記述されたデバイス・ドライバを介して，ハードウェアにアクセス)
6) 対応するOSごとに1)～4)の手順を繰り返す

● WinDriverによる開発手順

WinDriverを用いると，カーネル・モードの開発はもちろん，OSの内部構造やバス・プロトコルの知識の習得といった手順を省いても開発が行えるので，結果として，開発時間の短縮となります．

WinDriverでの開発手順は，WinDriverのDriverWizardを使用してハードウェアの動作を検証した後，自動的にクロス・

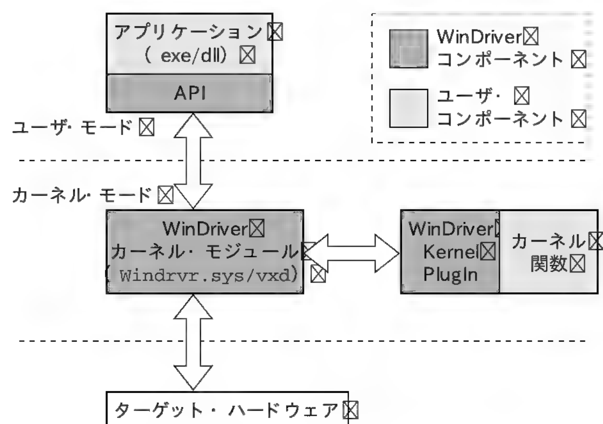


図1 WinDriverのアーキテクチャ

注1：デバイス・ドライバは，実行するOSと密接な関係があり，デバイス・ドライバを開発する際には，OSのアーキテクチャおよび内部構造の知識が必要となる．
注2：ドライバ・インターフェースのセットには，Windows 98/Me/NT/2000/XP/2003 ServerのDDK，Windows CEのETKなどがある．ドライバには，各OS独自のAPIを使用し，開発者はOSに応じたドライバを開発する必要がある．

プラットホームな(互換性のある)ドライバ・コードを生成します．それを雛形とし，必要な機能のみを追加するだけで開発できます．DriverWizardでは，一般的な開発環境(Microsoft Visual Studio，Borland Builder，Linux gmakeなど)用のメイク・ファイルを生成します．生成されたコードは修正なしにコンパイルし，実行できます．

2 WinDriverを使ったデバイス・ドライバの開発手順

ここからは，WinDriverを使用するとどれくらいの作業が削減できるのかを知るために，実際にWinDriverを使ってPCIカードのWindows用のデバイス・ドライバを開発していきます．

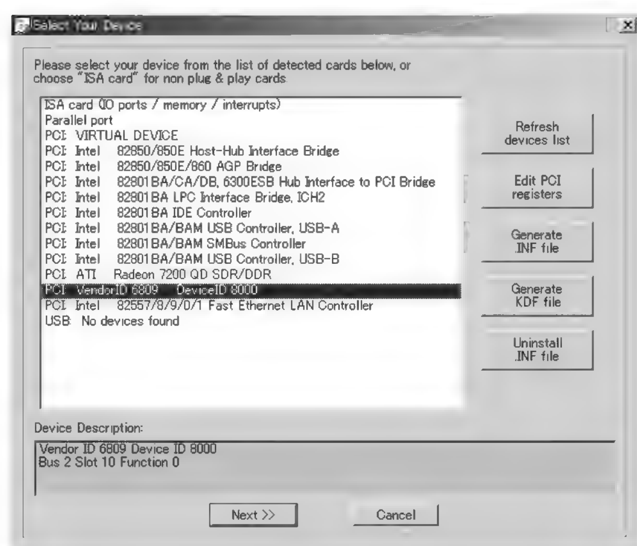


図2　対象となるPCIカードを選択

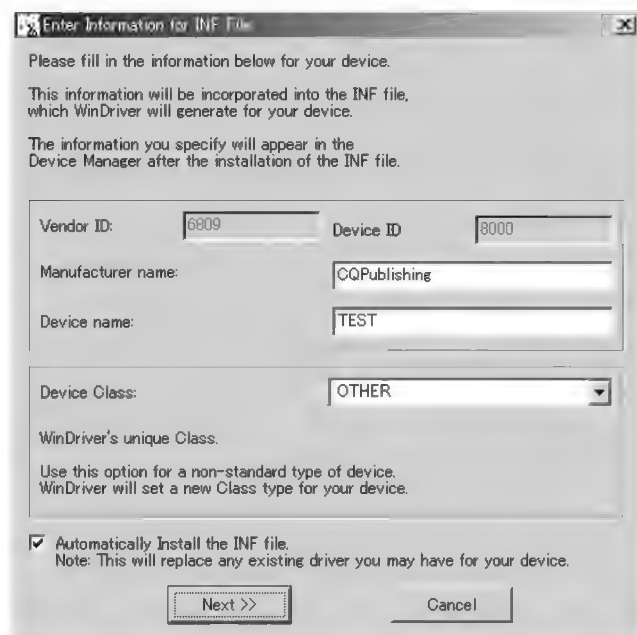


図3　INFファイルの生成

今回，ターゲットとしたPCIカード(VenderID 6809，Device ID 8000)の動作概要は，以下のとおりです．

- ディップ・スイッチ入力レジスタを読み出すと，PCIボード上に実装した8ビット・ディップ・スイッチの状態を読み出す．ディップ・スイッチの状態がONで'1'，OFFで'0'
- LED点灯データ出力レジスタに'1'を書き込むと，PCIボード上の8ビットLEDが点灯する
- PCIボード上のプッシュ・ボタンを押下すると，割り込みが発生する

● ステップ1 ── インストール

WinDriverのインストール・プログラム(`WDxxx.EXE`．`xxx`はバージョン番号)を起動し，インストーラの指示に従ってインストールを実行します．インストールは，システム管理者の権限のあるユーザで行ってください．

インストールの際には必要ありませんが，WinDriverで生成されたコードをコンパイルおよびビルドするためにC/C++，Visual Basic，Delphiなどの32ビット開発環境が必要となります．また，Kernel PlugIn機能を使用する場合にのみ，DDKをインストールする必要があります．

● ステップ2 ── ハードウェアの選択

次に，下記のような手順でハードウェアの選択を行います．

- DriverWizardを起動し，表示されたダイアログ・ボックスから[Create a new driver project]を選択
- [Select Your Device]ダイアログ・ボックスで，プラグ＆プレイ・カードがすべて表示されるので，ターゲットとなる[PCI Vendor ID 6809 Device ID 8000]を選択(図2)

● ステップ3 ── INFファイルの生成

PCIデバイスのドライバがインストールされていない新規のプラグ＆プレイ・カード用のドライバを開発する場合は，対象のデバイスのINFファイルを生成してインストールする必要があります．DriverWizardでは，これを自動的に行えます(図3)．

- [Generate .INF file]または[Next]をクリック
- 表示されたダイアログ・ボックスに必要な項目を入力
- [Next]をクリックし，INFファイルの保存先のディレクトリを選択．Windows 2000/XP/Server 2003上では，[Automatically Install INF file]オプションをオンにすることによってDriverWizardが自動的にINFファイルをインストールする
- INFファイルのインストールが終了したら，ステップ2の[Select Your Device]ダイアログ・ボックスに戻り，再度，対象のデバイスを選択

● ステップ4 ── ハードウェアの検出と定義

DriverWizardは，プラグ＆プレイ・ハードウェアのリソース(I/O範囲，メモリ範囲，割り込み)を自動的に検出します(図4)．

I/Oレジスタ(表1)については，手動で定義します．[Registers]タブをクリックし，[New]ボタンをクリックする



図4　対象となるPCIカードのリソースの検出



図5　レジスタの定義
Register0を定義．Register1～3も同様に定義する

と，レジスタが定義できます(図5)．

次に，割り込み情報の定義を行います(図6)．

● ステップ5──ハードウェアの検証

デバイス・ドライバを記述する前に，ハードウェアが期待どおりに機能するかどうかを診断する必要があります．ドライバのコードを記述することなく，DriverWizardでハードウェアの診断を行えます．

- I/O，メモリ，レジスタへの読み書きを行う
- Register0を検証する．ディップ・スイッチ入力レジスタを読み出す．PCIボード上の8ビット・ディップ・スイッチの状態がONでビットが1(図7)
- Register1～3についても，検証していく
- ハードウェアの割り込みをListen(確認)する．[Listen to Interrupts]をクリック(図8)

● ステップ6──ドライバ・コードの生成

ハードウェアの検証後，DriverWizardで自動的にドライバの雛型となるコードを生成します．

- [Next]をクリックするか，Buildメニューから[Generate Code]を選択
- [Select Code Generation Options]画面で，開発言語を選択し，

表1　対象となるPCIボードのI/Oレジスタの仕様

オフセット	アクセス・サイズ	R/W	ビット	用途
+00h	32ビット	R	ビット7～0	ディップ・スイッチ入力レジスタ
+04h	32ビット	R/W	ビット7～0	LED点灯データ出力レジスタ
+08h	32ビット	R	ビット0	割り込みステータス・レジスタ(‘1’で割り込み発生)
		W	ビット0	割り込み要求クリア(‘1’で割り込みクリア)
+0Ch	32ビット	R/W	ビット0	割り込みマスク・レジスタ(‘1’で割り込み解除)

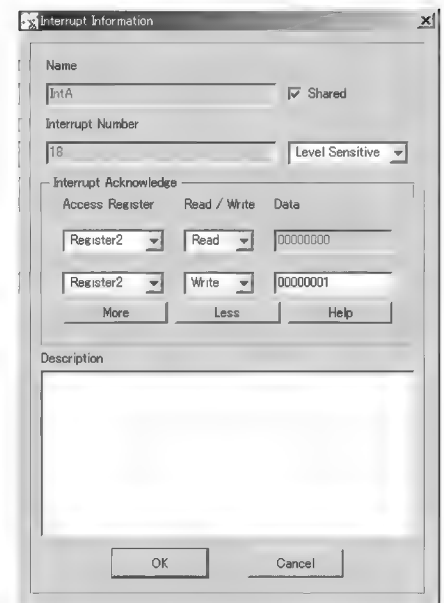


図6
割り込みの定義

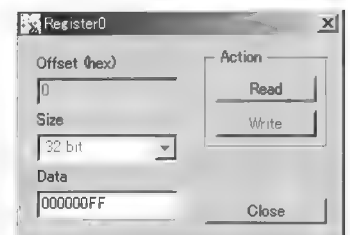


図7
ハードウェアの検証
Register0を検証．Register1～3も同様に検証する

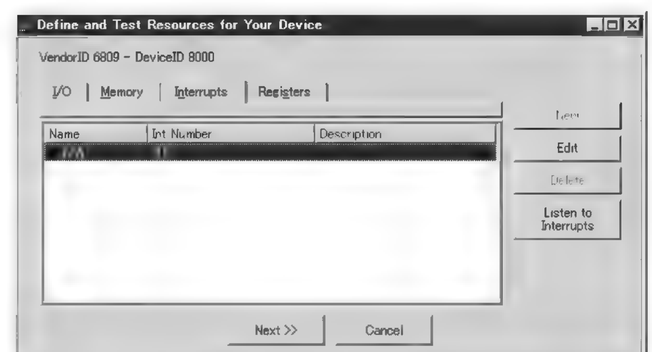
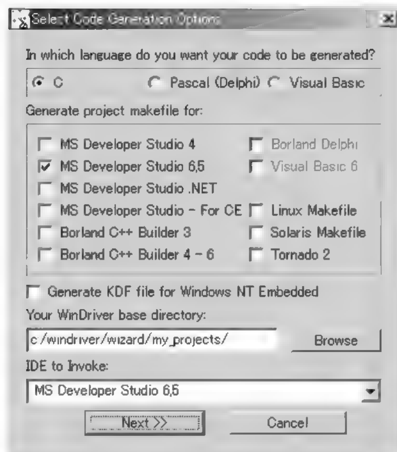
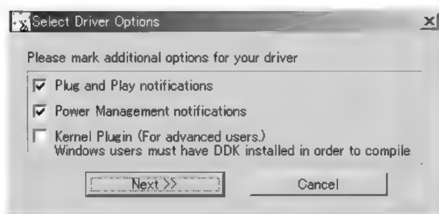


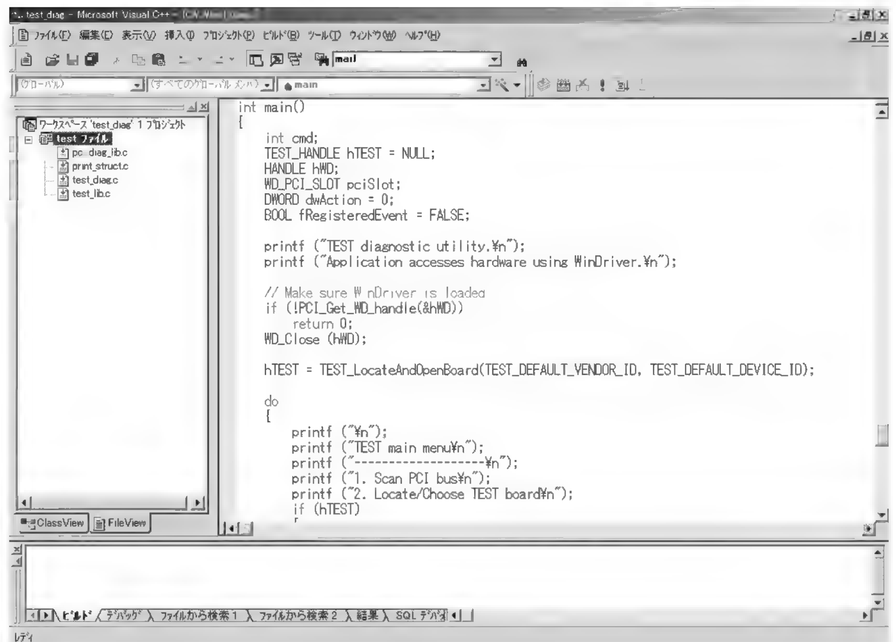
図8　割り込みの確認



(a) 開発環境の選択



(b) オプションの選択



(c) コンパイラの起動

図9　ドライバ・コードの生成

リスト1　DriverWizardが生成したAPI関数

```
// 関数: TEST_ReadRegister1()
//    レジスタ Register1 からの読み込み
// 引き数:
//    hTEST [in] - TEST_Open() 関数から受信したカードへのハンドル
// 戻り値:
//    レジスタから読み込んだ値

UINT32 TEST_ReadRegister1 (TEST_HANDLE hTEST)
{
    return TEST_ReadDword(hTEST, (TEST_ADDR)
               TEST_Register1_SPACE, TEST_Register1_OFFSET);
}
```

(a) Register1からの読み込み

```
// 関数: TEST_WriteRegister1()
//    レジスタ Register1 への書き込み
// 引き数:
//    hTEST [in] - TEST_Open() 関数から受信したカードへのハンドル
//    data [in] - レジスタへ書き込むデータ data [in]
// 戻り値:
//    なし

void TEST_WriteRegister1 (TEST_HANDLE hTEST, UINT32 data)
{
    TEST_WriteDword(hTEST, (TEST_ADDR) TEST_Register1_SPACE,
                           TEST_Register1_OFFSET, data);
}
```

(b) Register1への書き込み

作成するプロジェクトの開発環境を決める．ここでは，“C言語”で“MS Developer Studio 6.5”を選択した〔図9(a)〕

● ドライバ・コード内でPlug-and-PlayとPower Managementイベントを処理する場合と，Kernel PlugInコードを生成する場合には，図9(b)の画面で選択(Kernel PlugIn機能を使用する場合には，DDKをインストールする必要がある)

● [Next]をクリック．DriverWizardが自動的にコードを生成し，選択したコンパイラを起動する

C/C++を使用する場合，DriverWizardは次のファイルを生成します(“test”はプロジェクト名)．

● test_files.txt――生成されたファイルの説明を記述したReadmeファイル

● test_diag.c――ターゲット・デバイス用にDriverWizardが生成した雛形となるアプリケーション．ライブラリ関数の使用方法を明示する

● test_lib.c――test_diag.cで使用するデバイスへのアクセスで使用するユーティリティ関数の一般的なライブラリ

● test_lib.h――ユーティリティ関数のヘッダ・ファイル

● 選択した開発環境用のプロジェクト・ファイル

DriverWizardが生成したコードをビルドし，実行します．これでPCIカードの診断アプリケーションの完成です．この診断アプリケーションがドライバの雛型となります．

● DriverWizardが生成したコード

リスト1のコードは，DriverWizardが生成したAPI関数です．ステップ4で定義したリソースBar0 Rangeにあるレジスタ“Register1”にアクセスし，Read/Writeするためのコードです．

なお，コメント文も自動的に生成されますが，英語で書かれているため，ここでは日本語に訳して掲載しています．

そのほか，PCIカードをWinDriverのカーネル・モジュールと動作するようにレジスト/アンレジストするためのコードと，

表2　おもなWinDriver APIの一覧

開始&終了		WD_Open() WD_Close() WD_Version() WD_License()
PCI		WD_CardRegister() WD_CardUnregister() WD_CardCleanupSetup() WD_PciScanCards() WD_PciGetCardInfo() WD_PciConfigDump() WD_IsapnpScanCards() WD_IsapnpGetCardInfo() WD_IsapnpConfigDump()
PCIのI/Oおよびメモリ・アクセス		WD_Transfer() WD_MultiTransfer() PCI DMA WD_DMALock() WD_DMAUnlock()
PCI割り込み処理		InterruptThreadEnable() InterruptThreadDisable()
	低レベル	WD_IntEnable() WD_IntDisable() WD_IntCount() WD_IntWait()
Plug-and-Play&パワー・マネージメント		WD_EventRegister() WD_EventUnregister() WD_EventPull() WD_EventSend()
ユーティリティ&デバッグ		WD_Debug() WD_DebugAdd() WD_DebugDump() WD_LogStart() WD_LogStop() WD_LogAdd() WD_Sleep()

PCIカードの割り込みを有効/無効にするコードについては，長くなるのでここでは割愛します．本誌のWebサイト(http://www.cqpub.co.jp/interface/download/contents.htm)に掲載するので，参照してください．

おもなWinDriver APIの一覧を表2に示します．

3 デバッグとパフォーマンスの向上，互換性

● デバッグ

従来のデバイス・ドライバのデバッグでは，カーネル・モードでのデバッグが必要でした．デバイス・ドライバは，OSの一部として動作するので，デバッガによってドライバのプロセスを停止するとOSも停止し，ほかのプロセスも停止してしまいます．そのため，デバイス・ドライバをデバッグする際には，2台のコンピュータをケーブルでつなぎ，1台をデバッグするソフトウェアを実行するホストとし，もう1台をターゲットのマシンとしてデバッグを行います．

WinDriverではユーザ・モードでドライバを開発するので，開発環境マシン上でユーザ・モードのデバッグ・ツールを使用してデバッグを行います．WinDriverにはデバッグ・ツールとしてデバッグ・モニタ・ユーティリティがあり(GUIベースとコンソール・モードがある)，デバッグ・モニタでWinDriverのカーネル(windrvr.sys/windrvr.vxd/windrvr.dll/windrvr.o/wdpnp.sys)が処理するすべての動作を監視し，カーネルへ送られる各コマンドがどのように実行されるかを監視できます(図10)．WinDriverで作成するドライバは，ユーザ・モードで動作するため，MS Developer Studioのユーザ・モード・デバッガも使用できます．

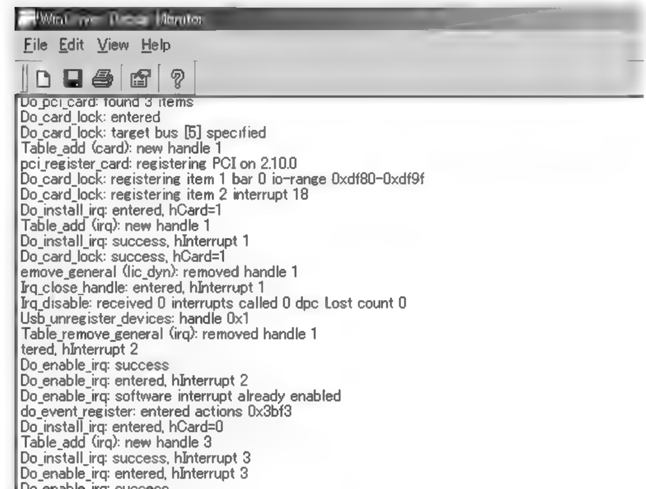


図10　デバッグ・モニタの出力内容

● パフォーマンスの向上

アプリケーション・レベルでメモリおよび割り込みを処理する際に，カーネルからユーザ・モードへの関数を呼び出すところでオーバヘッドが発生します．この問題を解決するには，パフォーマンスの重要なコード部分をカーネル・レベルで実行できるアーキテクチャが必要です．開発者は，最初にユーザ・モードで簡単にすばやく開発し，パフォーマンスの重要な部分のコードを必要に応じて切りわけます．WinDriverには，Kernel PlugInアーキテクチャがあり，これでコードのパフォーマンスの重要な部分をユーザ・モードからカーネル・モードに移行し，コードのパフォーマンスの最適化を行えます．ただし，Kernel PlugInを使用する際には，DDKをインストールする必要があります．

WinDriverは，メモリ転送コマンドをカーネル・レベルで実行するので，通常はKernel PlugInを使用しません．Kernel PlugInを必要とするのは，ハードウェアが高い割り込み速度を必要とする場合や，ハードウェアのメモリがメモリにマップされていない場合(たとえばI/Oマップ)などです．PCIデバイスの場合，簡単なハードウェアの修正でハードウェアをI/OマップからメモリＥ・マップに変更できます．メモリ・マップのカードの場合，WinDriverはユーザ・モードのポインタを提供し，これを使用してユーザ・モードから直接カードのメモリから

WinDriverのユーザ事例

COLUMN

本文中ではPCIデバイス・ドライバの開発手法を紹介しましたが，WinDriverには，USB1.1/2.0に対応したバージョンもあります．実際にWinDriverを使用して，USBのデバイス・ドライバ開発を行ったユーザの事例を紹介します．

● 製品概要――MSXゲームリーダー

アスキーおよびマイクロソフトが提唱した8ビットのゲーム機「MSX」向けのゲームROMカートリッジをWindowsが搭載されたPCよりアクセスするUSB機器です．公式エミュレータである「MSXPLAYer」とセットで使用し，Windows上でMSXのソフトを動作させることができます．

● WinDriverの使用

ルネサステクノロジ製「H8S/2215UF」を使用したUSBデバイスである「MSX ゲームリーダー」のドライバDLLの作成にWinDriverを使用しました．前述の公式エミュレータから，作成したドライバにアクセスしています．処理としては，ゲームROMカートリッジにアクセスするためのアドレスとデータの受け渡しと，書き込み/読み出しデータの転送を行っています．

● 開発環境

- Windows 2000 Professional
- Visual Studio C++6.0（ドライバ側のみ．エミュレータ部は別環境で開発して組み合わせた）

● WinDriverを採用した理由

- 動作が安定していること
- 学習が容易なこと
- 開発効率が良いこと
- デバイスの着脱に対応していること

● WinDriverのメリット

- ウィザードに従って設定するだけで，INFファイルとVC++での雛形を含んだプロジェクトが作成され，その時点でデバイスのテストを行うことができた
- USBデバイスの着脱に対応していた．デバイスのアクセス中にデバイスが抜かれた場合も適切な処理がなされた（アクセス関数が処理を中断し，エラー・コードを返した．着脱時の処理はコールバック関数として記述するだけ）
- Windows APIの`ReadFile/WriteFile`関数を使うような感覚で，特に違和感なくプログラミングできた
- ユーザ・モードで開発し，デバッグできたため開発効率が良かった
- 評価版があり，かつ評価版の試用期間を延長できたので，十分な評価を行ったうえで採用できた

データを転送でき，`WD_Transfer()` API関数を呼ぶ必要がなく，かつパフォーマンスを向上します．

●クロス・プラットホーム――ドライバ・コードの互換性

WinDriverは，ハードウェアにアクセスするアプリケーション・レベルのAPIを提供します．そのAPIは，カーネル・モジュールを呼び出し，OS 独自のカーネルAPIを使用してハードウェアにアクセスします．さまざまなOSのカーネル・モジュールを提供することによって，コードの修正をせずにドライバをほかのOSに移植できます．

WinDriverで開発したドライバは，対応するOS（Windows 98/Me/NT/2000/XP/Server 2003/CE，Linux，Solaris，VxWorks）間でソース・コード・レベルで互換性があります．このうち，Windowsの間ではバイナリ・レベルで互換性があります．UNIXシステムの場合，ソースでの互換性があるので，再コンパイルのみが必要となります．WinDriverでは，生成したコードを修正せずに，ほかのOSへも移行できる柔軟性を持っているのです．

＊　　＊

フル機能を備えた，機能制限のないWinDriverの評価版が，発売元のエクセルソフト（株）のWebサイトからダウンロードできるので，ぜひ，お試しください．

なお，製品版と評価版の違いは以下のようになっています．

- 評価版では，評価版であることを示すメッセージをつねに表示
- DriverWizardを使用中に評価版を実行していることを知らせるダイアログ・ボックスを表示
- Linux，Solaris，VxWorks，Windows CE版では，60分間動作した後，停止する．再度評価する際には，再ロードする必要がある

にし・のぶあき　XLsoft Corporation



プログラミングの要

宮坂 電人

第16回

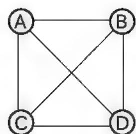
重みつきグラフ――距離や運賃の算出にも使えるアルゴリズム

今回も前回に引き続きグラフについて説明します．前回は辺に重みが付加されていないグラフについて説明しましたが，今回は重みつきグラフについて考えてみましょう．

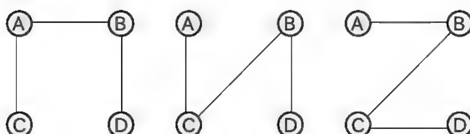
極大木と最小極大木

現実のプログラミングで重みつきグラフを取り扱うとき,「重み」とは具体的には2頂点間の距離であったり，移動時間や移動費用を意味します．たとえば，四つのノードでネットワークを配線するとしましょう．任意の二つのノードが必ず直結されるよう配線した場合には，6本のケーブルが必要です(図1)．このような任意の二つの頂点間で必ず辺が存在するグラフを「完全グラフ(complete graph)」と称します．

ところでこのとき，それぞれのノードがゲートウェイとして機能し，必ずしもノードどうしが直結されていなくてもよいなら，6本より少ない配線でもかまいません．図1で示した四角形状のグラフだと最低3本あれば機能します．このように任意の頂点が直結，あるいは別の頂点を経由して結線されている状態で(つまり「連結グラフ」である)，そこから最少本数の辺をピックアップして作成したツリーを「極大木(spanning tree)」と呼びます[注1]．さらに極大木のうち，辺の重みの合計がもっとも少ないものを「最小極大木(minimum spanningt tree, MSTと略称することがある)」あるいは「最小木」と称します．



四つの頂点を持つ完全グラフ⊠
任意の二つの頂点間に必ず辺がある⊠



極大木⊠
最少本数の辺でできたツリー⊠

図1
完全グラフと
極大木

最小極大木を求める手法はいくつかありますが，本連載で参考にしている「Mastering Algorithms with C」[注2]では，もっとも単純と思われる手法を紹介しています．グラフの任意の頂点を選び，そこから幅優先探索に似たグラフ巡回処理を行いますが，このとき頂点を記録したキューから，もっとも重みが少ない辺でつながっている頂点を優先して選びます．

そのために必要なことは，頂点を示すデータに，

- 隣接する頂点への距離
- 隣接する頂点へのポインタ

を付加することです．具体的にはリスト1で示すようなMSTVertex型を利用します．この型をそのまま使うか，これを継承した型を用意します．

辺については最小極大木専用の型は必要なく，重みが用意されていればいいでしょう．具体的にはリスト2で示すようなMSTEdge型を利用します．

グラフ巡回処理は前回紹介した幅優先探索と似ていますが，違いは頂点記録のキュー(aVertexQueue)から頂点を取り出すときに，キューの先頭からではなく，記録されている頂点のうち，隣接頂点からの距離(distanceメンバ)が最小であるものを取り出す点です．こうすることで最短距離の経路が優先して作成されるようになります．このあたりのロジックは，こと

リスト1　MSTVertex

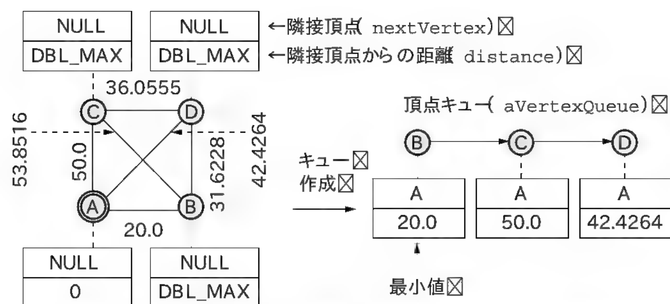
```
struct MSTVertex : public Vertex {
    double distance;          //隣接頂点からの距離
    MSTVertex* nextVertex;  //隣接頂点
};
```

リスト2　MSTEdge

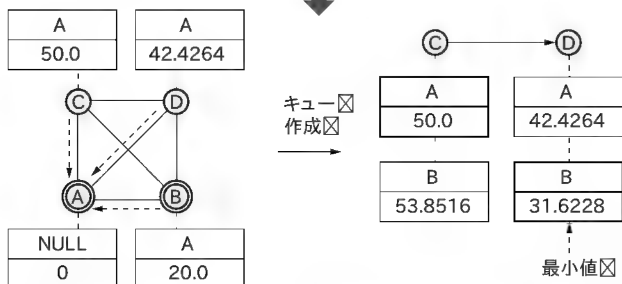
```
template <class VertexT>
struct MSTEdge : public Edge<VertexT> {
    double weight; //辺の長さ
};
```

注1：最少本数をピックアップしてグラフを作ると閉路は生じない．前回説明したとおり，閉路のないグラフは「ツリー」であるので，あえて「極大グラフ」と称していないのであろう．

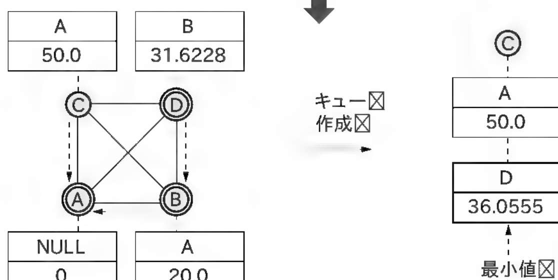
注2：http://www.oreilly.com/catalog/masteralgoc/を参照．

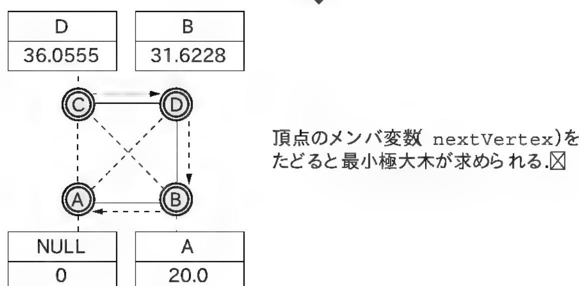


最小極大木の作成開始頂点(A)に隣接する頂点(B, C, D)との距離を求めながら頂点キューを作成する.
頂点キューから最小値の距離を持つ頂点(ここではB)を取り出し，次の巡回頂点とする.



Bに隣接する未巡回頂点(C, D)との距離を求め，その結果と頂点のメンバ変数を比較する.
すでに保持している距離と比較して，小さいほうを採用する.
Cのメンバ変数は更新されないが，Dは更新される.
頂点キューから最小値の距離を持つ頂点(ここではD)を取り出し，次の巡回頂点とする.





頂点のメンバ変数(nextVertex)をたどると最小極大木が求められる.

図2　最小極大木の生成

ばでの説明よりも実際にプログラムを追跡し，頂点のメンバ変数と頂点記録のキューがどのように変化するかを自分で絵を描いてみると理解しやすいかと思います(図2).

最小極大木を作成するクラスはリスト3のようになります．

startメンバ関数で頂点のメンバ変数を書き換え，どのように最小極大木を描くべきかをメンバ変数上に残すようにします．endメンバ関数によって頂点のメンバ変数を追跡し，最小極大木の経路を取得できます．MSTクラスを使った例はリスト4のようになります．

最短経路を求める

重みつきグラフを扱うプログラムでもっとも取り上げられる機会が多いのが，2点間で最短となる(あるいはもっとも運賃が安くなる)経路を求めるプログラムでしょう．さきほどの最小極大木を求めるプログラムでは，もっとも短い配線を求めることはできても，最短距離を選ぶ助けにはなりません．なぜなら，最小極大木を求めるということは，全体が最短となる経路を求める処理であって，任意の2点間の最短経路を求める処理にはならないからです．たとえば，最短で直結する辺があるにもかかわらず，極大木に沿っていないことがあるからです．また2点間が直結されていても，そこが最短距離とは限らず，迂回したほうが最短になる場合もあります．

「Mastering Algorithms with C」では最短距離を求める手法として有名な「ダイクストラ法」を採用しています．文字通り，ダイクストラ氏[注3]が考案した手法です．これはおもしろいことに，さきほどの最小極大木を求める手法と似通っています．違いは，頂点に記録する隣接頂点への距離が，始点から各頂点

リスト4　MSTクラスの使用例

```
struct MyVertex : public MSTVertex {
    int x,y; //座標
    bool equalVertex(const Vertex* iVertex) const { //同値判定
        const MyVertex* aVtx
          = dynamic_cast<const MyVertex*>(iVertex);
        return (aVtx != NULL) && (aVtx->x == x)
                 && (aVtx->y == y);
    }
    MyVertex(int iX = 0,int iY = 0) {
        x = iX;
        y = iY;
    }
};

typedef MSTEdge<MyVertex> MyEdge;
typedef Graph<MyVertex,MyEdge> MyGraph;

#define ARRAY_SIZE(X) (sizeof(X) / sizeof(X[0]))

//グラフの初期化
static void initMyGraph(MyGraph& iGraph)
{
    unsigned int aI,aJ;
    typedef struct {
        int x;
        int y;
    } Cood_t;
    static Cood_t aCood[] = {
```

注3：Edsger W. Dijkstra(1930年～2002年)，構造化プログラミングの提唱者として有名なコンピュータ科学者．詳細な履歴についてはhttp://www.cs.utexas.edu/users/EWD/に詳しい．

リスト 3　MSTクラス

```
template <class VertexT,class EdgeT>
class MST {
public:
    typedef Graph<VertexT,EdgeT> Graph_t;
private:
    Graph_t& mGraph; //作成対象のグラフ
    MST(); /* (empty) */
public:
    //コンストラクタ
    MST(Graph_t& iGraph) : mGraph(iGraph) { /* (empty) */ }

    //MSTを求める処理のメイン
    //iStartVertex=巡回開始点
    void start(const VertexT& iStartVertex){
        //隣接情報の全リスト
        typename Graph_t::AdjList& aAdjList = mGraph.adjList();
        //隣接情報へのイテレータの型
        typedef typename Graph_t::AdjList::iterator AdjIterator;
        //隣接情報へのイテレータ
        AdjIterator aAdjItr = aAdjList.end();
        //全頂点のフラグを消去する
        for(AdjIterator aItr = aAdjList.begin();
              aItr != aAdjList.end(); aItr++){
            VertexT* aVtx = aItr->startVertex;
            aVtx->visited = false;
            aVtx->nextVertex = NULL;
            //この頂点が始点であるかを判断する
            if(iStartVertex.equalVertex(aVtx)){
                //始点であるなら
                aVtx->distance = 0;
                aAdjItr = aItr;
            }else{ //始点でないなら
                aVtx->distance = DBL_MAX;
            }
        }
        //始点が見つからなかったなら処理が続行できないので戻る
        if(aAdjItr == aAdjList.end())
            return;
        //最短隣接判断用の頂点キュー
        std::list<VertexT*> aVertexQueue;
        //始点を頂点キューに格納する
        aVertexQueue.push_back(aAdjItr->startVertex);
        //頂点キューが空にならない限り処理を継続する
        while(!aVertexQueue.empty()){
            //距離が最小になる隣接頂点がある隣接情報を
            //イテレータ・キューから探す
            double aMinDistance = DBL_MAX;
            typename std::list<VertexT*>::iterator aVtxItr,
                                                   aMinVertexItr;
            for(aVtxItr = aVertexQueue.begin();
                  aVtxItr != aVertexQueue.end(); aVtxItr++){
                VertexT* aVtx = *aVtxItr;
                if(aMinDistance > aVtx->distance){
                    aMinDistance = aVtx->distance;
                    aMinVertexItr = aVtxItr;
                }
            }
            //求めた頂点の情報を保持し, イテレータ・キューから
            //削除する
            VertexT* aMinVertex = *aMinVertexItr;
            aVertexQueue.erase(aMinVertexItr);
            //求めた頂点を訪問ずみとする
            aMinVertex->visited = true;
            //求めた頂点の辺をたどる
            typename Graph_t::AdjList::iterator aMinAdjCell
              = mGraph.adjCell(*aMinVertex);
            typename Graph_t::EdgeList& aMinEdgeList
              = aMinAdjCell->edges;
            typename Graph_t::EdgeList::iterator aEdgeItr;
            for(aEdgeItr = aMinEdgeList.begin();
                  aEdgeItr != aMinEdgeList.end(); aEdgeItr++){
                //未巡回であるなら処理対象とする
                VertexT* aEVtx = aEdgeItr->endVertex;
                if(!aEVtx->visited){
                    //隣接頂点がNULLなら頂点キューに登録する
                    if(aEVtx->nextVertex == NULL)
                        aVertexQueue.push_back(aEVtx);
                    //隣接頂点が保持する最短距離を
                    //更新できるなら, 内容を書き換える
                    if(aEdgeItr->weight < aEVtx->distance){
                        aEVtx->distance = aEdgeItr->weight;
                        aEVtx->nextVertex = aMinVertex;
                    }
                }
            }
        }
    }

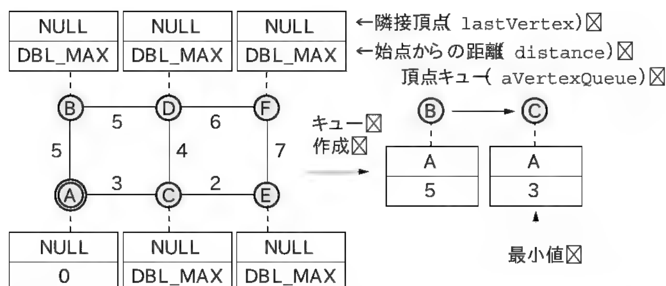
    //startで求めた結果からMSTを求める
    //oPath=経路を格納するリスト(頂点のペアを連続して格納する)
    void end(std::list<VertexT*>& oPath){
        typename Graph_t::AdjList& aAdjList
          = mGraph.adjList(); //隣接情報の全リスト
        //隣接情報へのイテレータ
        typename Graph_t::AdjList::iterator aItr;
        //隣接頂点を持っている頂点から情報を引き出す
        for(aItr = aAdjList.begin(); aItr != aAdjList.end();
              aItr++){
            VertexT* aVtx = aItr->startVertex;
            if(aVtx->visited && aVtx->nextVertex != NULL){
                oPath.push_front(aVtx);
                oPath.push_front(static_cast<VertexT*>(
                  aVtx->nextVertex));
            }
        }
    }
};
```

リスト 4　MSTクラスの使用例(つづき)

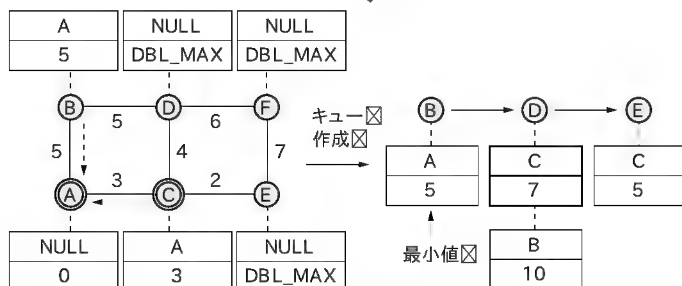
```
        {0,0},{0,6},{2,3},{2,5},{3,1},{4,5},
        {4,7},{5,3},{6,1},{7,5},{7,7}
    };
    for(aI = 0; aI < ARRAY_SIZE(aCood); aI++){
        MyVertex aVtx(aCood[aI].x,aCood[aI].y);
        iGraph.insertVertex(aVtx);
        for(aJ = 0; aJ < aI; aJ++){
            MyVertex aEVtx(aCood[aJ].x,aCood[aJ].y);
            MyEdge aEdge;
            double aXdiff = static_cast<double>(aCood[aI].x
                                                - aCood[aJ].x);
            double aYdiff = static_cast<double>(aCood[aI].y
                                                - aCood[aJ].y);
            aEdge.weight = std::sqrt(aXdiff * aXdiff
                                     + aYdiff * aYdiff);
            iGraph.insertEdgeUD(aVtx,aEVtx,aEdge);
        }
    }
}

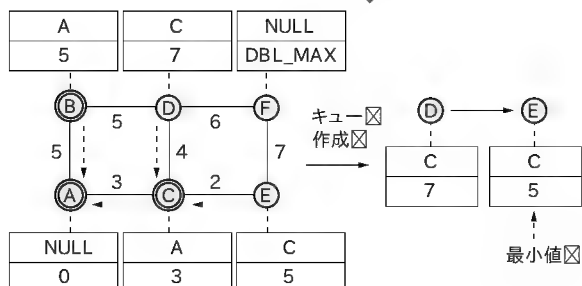
//デモ
static void demo()
{
    MyGraph aGraph;

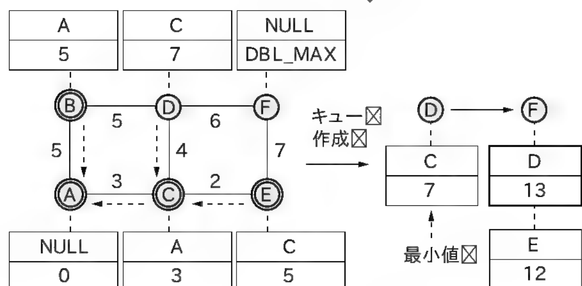
    //グラフの初期化
    initMyGraph(aGraph);
    //MST作成開始
    MST<MyVertex,MyEdge> aMSTMaker(aGraph);
    MyVertex aStartV(0,0);
    aMSTMaker.start(aStartV);
    //MSTの取得
    std::list<MyVertex*> aPath;
    aMSTMaker.end(aPath);
    //MSTの表示
    std::cout << "* dump MST start *¥n";
    while(!aPath.empty()){
        MyVertex* aVtx = aPath.front();
        std::cout << "vertex(" << aVtx->x << ","
                  << aVtx->y << ") - vertex(";
        aPath.pop_front();
        aVtx = aPath.front();
        std::cout << aVtx->x << "," << aVtx->y << ")¥n";
        aPath.pop_front();
    }
    std::cout << "* dump MST end *¥n";
}
```

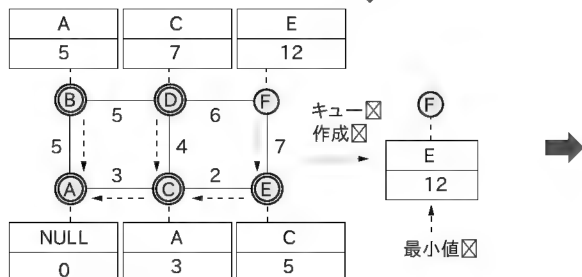


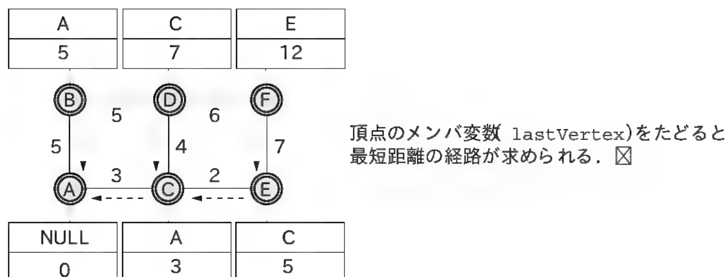
始点(A)に隣接する頂点(B，C)との距離を求めながら頂点キューを作成する.☒
頂点キューから最小値の距離を持つ頂点(ここではC)を取り出し，次の巡回頂点とする．☒











頂点のメンバ変数(lastVertex)をたどると最短距離の経路が求められる．☒

図3　最短距離を求める処理

リスト5　ShortestVertex

```
struct ShortestVertex : public Vertex {
    double distance;            //始点からの距離(経由した合計値)
    ShortestVertex* lastVertex; //始点もしくは始点に通じる頂点
};
```

リスト6　ShortestEdge

```
template <class VertexT>
struct ShortestEdge : public Edge<VertexT> {
    double weight; //辺の長さ
};
```

を経由した距離になっているところだけです．リスト5のような頂点の型を利用します．この型をそのまま使うか，これを継承した型を用意します．

辺については最小極大木のときと同様，専用の型は必要なく，重みが用意されていればいいでしょう．具体的にはリスト6で示すようなShortestEdge型を利用します．

同じく，グラフ巡回処理は前回紹介した幅優先探索と似ていますが，頂点記録のキュー(aVertexQueue)から頂点を取り出すときに，キューの先頭からではなく，記録されている頂点のうち，始点からの距離(distanceメンバ)が最小であるものを取り出す点です．このあたりのロジックも実際にプログラム・コードを追跡し，頂点のメンバ変数と頂点記録のキューがどのように変化するかを自分で絵を描いてみると理解しやすいかと思います(図3)．

最短経路を求めるクラスはリスト7のようになります．startメンバ関数で頂点のメンバ変数を書き換え，どのように最短経路をたどるべきかをメンバ変数上に残すようにします．endメンバ関数によって頂点のメンバ変数を追跡し，最短経路を取得できます．ShortestSearchクラスを使った例はリスト8のようになります．

閉路検出と訪問判断の3値化

前回，トポロジカル・ソートで閉路の検出は案外めんどうといった件です．

リスト 7　ShortestSearchクラス

```
//最短経路探索クラス
template <class VertexT,class EdgeT>
class ShortestSearch {
public:
    typedef Graph<VertexT,EdgeT> Graph_t;
private:
    Graph_t& mGraph; //探索対象のグラフ
    ShortestSearch(); /* (empty) */
public:
    //コンストラクタ
    ShortestSearch(Graph_t& iGraph) : mGraph(iGraph) {
                                        /*(empty)*/ }

    //最短距離を求める処理のメイン
    //iStartVertex=巡回開始点
    void start(const VertexT& iStartVertex){
        //隣接情報の全リスト
        typename Graph_t::AdjList& aAdjList = mGraph.adjList();
        //隣接情報へのイテレータの型
        typedef typename Graph_t::AdjList::iterator AdjIterator;
        //隣接情報へのイテレータ
        AdjIterator aAdjItr = aAdjList.end();
        //全頂点のフラグを消去する
        for(AdjIterator aItr = aAdjList.begin();
              aItr != aAdjList.end(); aItr++){
            VertexT* aVtx = aItr->startVertex;
            aVtx->visited = false;
            aVtx->lastVertex = NULL;
            //この頂点が始点であるかを判断する
            if(iStartVertex.equalVertex(aVtx)){
                //始点であるなら
                aVtx->distance = 0;
                aAdjItr = aItr;
            }else{ //始点でないなら
                aVtx->distance = DBL_MAX;
            }
        }
        //始点が見つからなかったなら処理が続行できないので戻る
        if(aAdjItr == aAdjList.end())
            return;
        //最短距離判断用の頂点キュー
        std::list<VertexT*> aVertexQueue;
        //始点を頂点キューに格納する
        aVertexQueue.push_back(aAdjItr->startVertex);
        //頂点キューが空にならない限り処理を継続する
        while(!aVertexQueue.empty()){
            //始点からの距離が最小になる頂点がある隣接情報を
            //イテレータ・キューから探す
            double aMinDistance = DBL_MAX;
            typename std::list<VertexT*>::iterator aVtxItr,
                                                aMinVertexItr;
            for(aVtxItr = aVertexQueue.begin();
                  aVtxItr != aVertexQueue.end(); aVtxItr++){
                VertexT* aVtx = *aVtxItr;
                if(aMinDistance > aVtx->distance){
                    aMinDistance = aVtx->distance;
                    aMinVertexItr = aVtxItr;
                }
            }
            //求めた頂点の情報を保持し，イテレータ・キューから
            //削除する
            VertexT* aMinVertex = *aMinVertexItr;
            aVertexQueue.erase(aMinVertexItr);
            //求めた頂点を訪問ずみとする
            aMinVertex->visited = true;
            //求めた頂点の辺をたどる
            typename Graph_t::AdjList::iterator aMinAdjCell
              = mGraph.adjCell(*aMinVertex);
            typename Graph_t::EdgeList& aMinEdgeList
              = aMinAdjCell->edges;
            typename Graph_t::EdgeList::iterator aEdgeItr;
            for(aEdgeItr = aMinEdgeList.begin();
                  aEdgeItr != aMinEdgeList.end(); aEdgeItr++){
                //未巡回であるなら処理対象とする
                VertexT* aEVtx = aEdgeItr->endVertex;
                if(!aEVtx->visited){
                    //始点もしくは始点に通じる頂点がNULLなら
                    //頂点キューに登録する
                    if(aEVtx->lastVertex == NULL)
                        aVertexQueue.push_back(aEVtx);
                    //求めた頂点の始点からの距離＋辺の長さを
                    //求める
                    double aNewDistance = aMinVertex->distance
                                          + aEdgeItr->weight;
                    //今求めた長さのほうが，終点にある distance
                    //より小さいなら，内容を書き換える
                    if(aNewDistance < aEVtx->distance){
                        aEVtx->distance = aNewDistance;
                        aEVtx->lastVertex = aMinVertex;
                    }
                }
            }
        }
    }

    //startで求めた結果から経路を求める
    //iEndVertex=巡回終了点,oPath=経路を格納するリスト
    void end(const VertexT& iEndVertex,std::list<VertexT*>&
                                                    oPath){
        VertexT* aVtx = mGraph.vertexPtr(iEndVertex);
        while(aVtx != NULL && aVtx->visited){
            oPath.push_front(aVtx);
            aVtx = static_cast<VertexT*>(aVtx->lastVertex);
        }
    }
};
```

リスト 8　ShortestSearchクラスの使用例

```
struct MyVertex : public ShortestVertex {
    std::string name; //識別用
    bool equalVertex(const Vertex* iVertex) const { //同値判定
        const MyVertex* aVtx = dynamic_cast<const MyVertex*>(
                                                    iVertex);
        return (aVtx != NULL) && (aVtx->name == name);
    }
    MyVertex(const char* iName) : name(iName) { /*(empty)*/ }
};

typedef ShortestEdge<MyVertex> MyEdge;
typedef Graph<MyVertex,MyEdge> MyGraph;

//グラフの初期化
static void initVEV(MyGraph& iGraph,const char* iV1,
  double iWeight,const char* iV2)
{
    MyVertex aV1(iV1);
    MyVertex aV2(iV2);
    iGraph.insertVertex(aV1);
    iGraph.insertVertex(aV2);
    MyEdge aEdge;
    aEdge.weight = iWeight;
    iGraph.insertEdge(aV1,aV2,aEdge);
}

static void initMyGraph(MyGraph& iGraph)
{
    initVEV(iGraph,"A",5.0,"B");
    ...(略)...
}

//デモ
static void demo()
{
    MyGraph aGraph;

    //グラフの初期化
    initMyGraph(aGraph);
    //探索開始
    ShortestSearch<MyVertex,MyEdge> aSS(aGraph);
    MyVertex aStartV("A");
    aSS.start(aStartV);
    //経路の取得
    std::list<MyVertex*> aPath;
    MyVertex aEndV("F");
    aSS.end(aEndV,aPath);
    std::cout << "* dump path start *¥n";
    for(std::list<MyVertex*>::iterator aItr = aPath.begin();
          aItr != aPath.end(); aItr++){
        MyVertex* aVtx = *aItr;
        std::cout << " [" << aVtx->name << "]";
    }
    std::cout << "¥n* dump path end *¥n";
}
```

ある頂点から辺をたどっていって元の頂点に戻るなら閉路になっているので，単純で力まかせの手法ではありますが，全頂点を始点とした深さ優先探索，あるいは幅優先探索を行い，巡回した頂点の辺を調べて始点に通じる辺を見つければいいように思います．しかし，この方法は頂点や辺の数が多くなるにつれて急速に処理時間が増えるため，実用的ではありません．すでに巡回した頂点であれば訪問フラグが`true`になっているので，巡回中の頂点の辺を調べて訪問フラグが`true`になっている頂点があれば閉路と判断していいように思います．

しかし，この方法は有向グラフの場合はあてになりません（図4）．図で示した例では元の頂点に戻る場合を閉路と判断することはもちろんのこと，合流地点となっている頂点が訪問済みになっていて閉路とまちがえることがあるからです．

実は有向グラフで閉路を検出する方法として，深さ優先探索を行い「深さ優先探索ツリー（depth first search tree，DFST と略称することがある）」を作る方法があります．深さ優先探索で頂点を巡回するとき，一つの頂点の辺を調べ，未巡回の頂点につながる辺をピックアップすると，それが深さ優先探索ツリーになるという理屈です．ちなみに，深さ優先探索ツリーにそっている辺を「木辺（tree edge）」と称します．木辺ではない辺には，

（1）ツリー上で親子関係の経路上にある頂点どうしがつながる辺
（2）ツリー上で親子関係の経路上にない頂点どうしがつながる辺

がありますが，（1）がある場合，閉路になっていると判断できます．ちなみに（1）の辺を「後退辺（back edge）」と称します[注4]．

木辺を判断するのはたいして難しくないのですが，親子関係の経路上であるかを調べるのはめんどうそうです．しかし，訪問フラグとは別のフラグ（その頂点のすべての辺で深さ優先探索が終わったと判断するフラグ）を用意するか，あるいは訪問フラグを2値ではなく3値にすることで簡単に判断できるようになります．3値とは具体的には，

- 白色――未訪問の頂点である
- 灰色――訪問した頂点であるが，そこにつながるすべての頂点が訪問済みではない
- 黒色――訪問した頂点であり，そこにつながるすべての頂点が訪問済みである

という意味です（図5）．

深さ優先探索をするとき，巡回中の頂点から白色の頂点につ

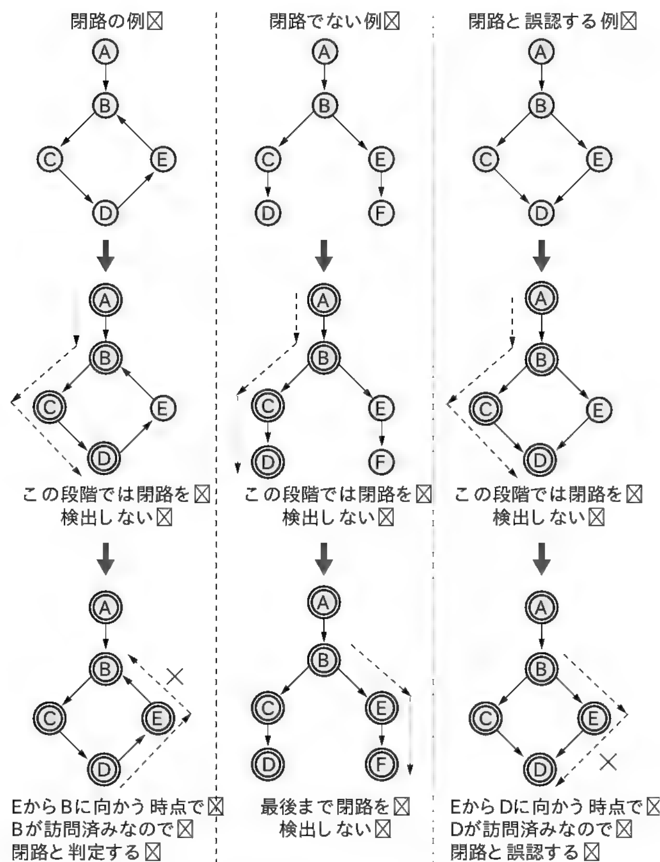


図4　訪問グラフによる閉路判定

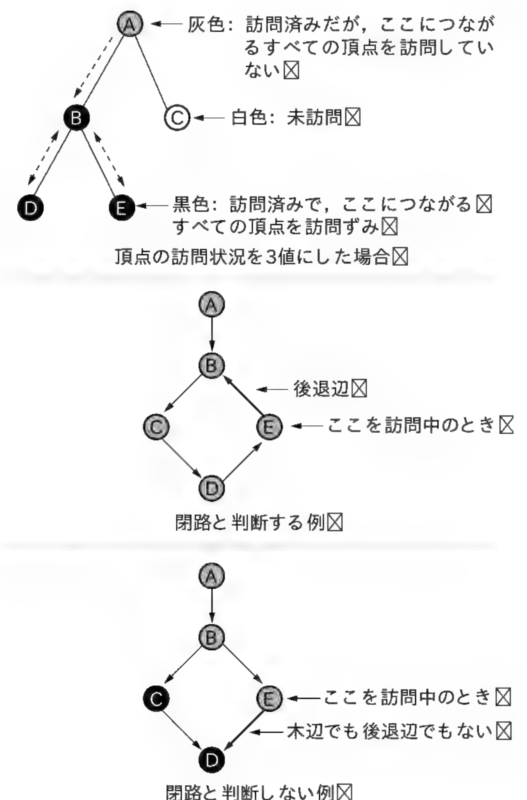


図5
頂点の色

注4：「逆辺」と翻訳している例もある．
注5：灰色の頂点は深さ優先探索ツリー上で親側となる頂点である．つまり子供から親側に向かって進む経路である．すでに親側から子供に向かって進む経路（木辺）があるので，この二つの経路が閉路となってしまう．

ながる辺は木辺であり，灰色の頂点につながる辺は後退辺になります[注5]．つまり，閉路検出は灰色の頂点につながる辺を見つければいいだけです．この3値の考えで頂点の型を書き直すとリスト 9のようになります．

深さ優先探索に使うコールバックのクラスはリスト 10のようになります．新たに増設したメンバ関数(forDFST)は深さ優先探索ツリーの作成や後退辺の検出に利用できます．

グラフ探索のクラスはリスト 11のようになります．また，GraphSearchクラスを使って深さ優先探索ツリーや後退辺を検出する例はリスト 12のようになります．

リスト 9　3値化した頂点の基本型

```
struct Vertex {
    enum color_t { WHITE,GRAY,BLACK }; //訪問状況を示す型
    color_t color; //訪問状況を示すフラグ
    //同値判定
    virtual bool equalVertex(const Vertex* iVertex) const = 0;
    virtual ~Vertex(){ /*(empty)*/ }
};
```

みやさか・でんと　miyadent@anet.ne.jp

リスト 10　GraphWalkerクラス

```
template <class GraphT,class VertexT,class EdgeT>
class GraphWalker { //グラフ探索コールバック用
public:
    //simpleWalkで全頂点に対して呼ばれる
    //iVertex=頂点情報,ioMisc=汎用ポインタ
    virtual void simpleVertex(typename GraphT::AdjList::iterator iVertex,void* ioMisc) { /*(empty)*/ }
    ...(略)...
    //depthFirstWalkで未巡回の頂点を発見したときに呼ばれる
    //iVertex=頂点情報,ioMisc=汎用ポインタ
    virtual void visitVertex(typename GraphT::AdjList::iterator iVertex,void* ioMisc) { /*(empty)*/ }

    //depthFirstWalkですべての終点の巡回をおえてメンバ関数から抜ける直前に呼ばれる
    //iVertex=頂点情報,ioMisc=汎用ポインタ
    virtual void afterVertex(typename GraphT::AdjList::iterator iVertex,void* ioMisc) { /*(empty)*/ }

    //depthFirstWalkの頂点処理で毎回呼ばれる
    //iVtx1=巡回中の始点,iVtx2=終点,ioMisc=汎用ポインタ
    virtual void forDFST(VertexT* iVtx1,VertexT* iVtx2,void* ioMisc) { /*(empty)*/ }
};
```

リスト 11　GraphSearchクラス

```
template <class VertexT,class EdgeT>
class GraphSearch {
public:
    typedef Graph<VertexT,EdgeT> Graph_t;
private:
    Graph_t& mGraph; //探索対象のグラフ
    GraphSearch(); /* (empty) */

    class ClearVisitedWalker : public GraphWalker<Graph_t,VertexT,EdgeT> { //全頂点を未訪問にするコールバック・オブジェクト用
    public:
        virtual void simpleVertex(typename Graph_t::AdjList::iterator iVertex,void* ioMisc) {
            VertexT* aVtx = iVertex->startVertex;
            aVtx->color = Vertex::WHITE;
        }
    };
public:
    //コンストラクタ
    GraphSearch(Graph_t& iGraph) : mGraph(iGraph) { /*(empty)*/ }

    //全頂点を未訪問にする
    void clearVisited() {
        ClearVisitedWalker aCVW;
        simpleWalk(aCVW);
    }

    //グラフの全頂点を単純に巡回する
    //iGraphWalker=コールバック・オブジェクト,ioMisc=汎用ポインタ
    void simpleWalk(GraphWalker<Graph_t,VertexT,EdgeT>& iGraphWalker,void* ioMisc = NULL){
        typename Graph_t::AdjList& aAdjList = mGraph.adjList();
        typename Graph_t::AdjList::iterator aItr;
        for(aItr = aAdjList.begin(); aItr != aAdjList.end(); aItr++){
            iGraphWalker.simpleVertex(aItr,ioMisc);
        }
    }

    //深さ優先探索(Depth First Search)をする
    //iStartVertex=巡回開始始点,iGraphWalker=コールバック・オブジェクト,ioMisc=汎用ポインタ
    void depthFirstWalk(const VertexT& iStartVertex,GraphWalker<Graph_t,VertexT,EdgeT>& iGraphWalker,void* ioMisc = NULL){
```

リスト 11　GraphSearchクラス(つづき)

```
        typedef typename Graph_t::AdjList::iterator DFWIterator;
        DFWIterator aItr; //巡回用イテレータ
        //巡回開始始点を探す
        aItr = mGraph.adjCell(iStartVertex);
        if(aItr == mGraph.adjList().end())
            return;
        //始点が巡回ずみなら戻る
        if(aItr->startVertex->color != Vertex::WHITE)
            return;
        //始点を巡回済みにする(ただし，この段階では灰色にする)
        aItr->startVertex->color = Vertex::GRAY;
        //始点を巡回させる
        iGraphWalker.visitVertex(aItr,ioMisc);
        //終点を巡回し，次の始点候補を探し，それを処理する
        typename Graph_t::EdgeList& aEList = aItr->edges;
        typename Graph_t::EdgeList::iterator aEItr;
        for(aEItr = aEList.begin(); aEItr != aEList.end(); aEItr++){
            VertexT* aVertex = aEItr->endVertex; //終点をえる
            iGraphWalker.forDFST(aItr->startVertex,aVertex,ioMisc); //DFST作成用
            if(aVertex->color == Vertex::WHITE) //未巡回の終点であるなら
                depthFirstWalk(*aVertex,iGraphWalker,ioMisc); //未巡回の終点を処理する
        }
        //始点を巡回済みにする(この段階で黒色にする)
        aItr->startVertex->color = Vertex::BLACK;
        //処理終了直前の頂点を知らせる
        iGraphWalker.afterVertex(aItr,ioMisc);
    }
    ...(略)...
};
```

リスト 12　main.cpp

```
struct SampleVertex : public Vertex {
    std::string name; //ノードの名前
    virtual bool equalVertex(const Vertex* iVertex) const {
        const SampleVertex* aVertex =
                    dynamic_cast<const SampleVertex*>(iVertex);
        return (aVertex != NULL) && (aVertex->name == name);
    }
    SampleVertex(const char* iName) : name(iName) { /*(empty)*/
                                                                   }
};

typedef Edge<SampleVertex> SampleEdge;
typedef Graph<SampleVertex,SampleEdge> SampleGraph;

static void initSub(SampleGraph& oGraph,const char* i1,
                                              const char* i2)
{
    SampleVertex a1(i1);
    SampleVertex a2(i2);
    oGraph.insertVertex(a1);
    oGraph.insertVertex(a2);
    oGraph.insertEdge(a1,a2);
}

//閉路ありの有向グラフ
static void initSampleGraph(SampleGraph& oGraph)
{
    initSub(oGraph,"A","B");
    initSub(oGraph,"B","C");
    initSub(oGraph,"C","D");
    initSub(oGraph,"D","E");
    initSub(oGraph,"E","B");
    initSub(oGraph,"D","F");
}

static std::list<SampleVertex*> gTreeEdge; //木辺
static std::list<SampleVertex*> gBackEdge; //後退辺

class MakeDFSTWalker : public GraphWalker<SampleGraph,
                              SampleVertex,SampleEdge> {
public:
    virtual void forDFST(SampleVertex* iVtx1,
                         SampleVertex* iVtx2,void* ioMisc) {
        //std::cout << "forDFST(" << iVtx1->name <<
                                "," << iVtx2->name << "),";
        //std::cout << "color=" << iVtx2->color << std::endl;
        switch(iVtx2->color){
        case Vertex::WHITE: //未巡回で木辺になる頂点
            gTreeEdge.push_back(iVtx1);
            gTreeEdge.push_back(iVtx2);
            break;
        case Vertex::GRAY:  //巡回ずみで後退辺になる頂点
            gBackEdge.push_back(iVtx1);
            gBackEdge.push_back(iVtx2);
            break;
        }
    }
};

static void dumpPath(std::list<SampleVertex*>& iPath)
{
    while(!iPath.empty()){
        SampleVertex* aVtx = iPath.front();
        std::cout << "vertex(" << aVtx->name << ") - vertex(";
        iPath.pop_front();
        aVtx = iPath.front();
        std::cout << aVtx->name << ")¥n";
        iPath.pop_front();
    }
}

static void demo()
{
    SampleGraph aGraph;
    //グラフの初期化
    initSampleGraph(aGraph);
    //DFST作成開始
    GraphSearch<SampleVertex,SampleEdge> aGS(aGraph);
    aGS.clearVisited();
    MakeDFSTWalker aWalker;
    SampleVertex aStartVtx("A");
    aGS.depthFirstWalk(aStartVtx,aWalker);
    //DFST表示
    std::cout << "* tree edge *¥n";
    dumpPath(gTreeEdge);
    std::cout << "* back edge *¥n";
    dumpPath(gBackEdge);
    std::cout << "* end *¥n";
}
```

組み込み向けリアルタイムLinuxディストリビューション

OCERA

(Open Components for Embedded Real-time Applications)

の概要

海老原 祐太郎

OCERA(http://www.ocera.org/)は，Universidad Politecnica de Valencia(スペイン バレンシア州立工芸大学)で開発が進められている組み込み向けのLinuxディストリビューションである．組み込みLinux向けとしては，オープン・ソース・プロジェクトの成果物や，各国の商用ベンダからいくつかのパッケージが提供されているが，今回は新たな選択肢としてOCERAプロジェクトを紹介する．　(筆者)

OCERAの紹介の前に，組み込みLinuxについてまとめてみましょう．広く組み込みLinuxと呼ばれていますが，一般のPC向けLinuxディストリビューションとの違いを表1にまとめます．

PC向けのLinuxは，サーバ用途やデスクトップ用途として使用されることを前提としているので，汎用的にアプリケーション・ソフトウェアをインストールして使用できるように考慮されています．また，最近のPCはとてもリッチなリソースを積んでいます．

一方で組み込みLinuxでは，特定の装置を構成するために最低限のコストで，CPU，メモリ，ストレージなどがとても限られていることが一般的です．一般的な組み込みLinuxでのリソースを表2に示します．

特にPC向けLinuxと比べてストレージ容量に厳しい制限があります．PC向けLinuxでは，数百Gバイトといった HDDが使用可能ですが，一般的な組み込みLinuxではMTD(本誌9月号参考)を利用して数Mバイトから大きくても128Mバイト程度のストレージしか使用できません．この限られたストレージ・スペースに，Linuxを構成するために必要なファイルを取捨選択して収めるよう設計されているLinuxを「組み込みLinux」と呼びます．

OCERAとは

OCERAは，Open Components for Embedded Real-time Applicationsの略で，リアルタイム・カーネルとmini-rootユーザ・ランドの双方を提供する，組み込みLinuxディストリビューションです．カーネルにはFSMLabs社のRTLinuxのGPL版を拡張したリアルタイムLinuxが使われています(ocera-1.0.0にはカーネルlinux-2.4.18のRTLinux拡張が含まれている)．

OCERAの特徴をつぎに示します．

- 組み込み向けに開発されているディストリビューション
- RTLinuxの拡張版カーネルを採用
- 多数のコンポーネント
- GPL，LGPLライセンス

ライセンスに関しては最後に詳しく述べます．OCERAプロジェクトの目標は，組み込み装置向けのリアルタイムAPIをサポートしたLinuxシステムを提供することです．

Linuxはカーネルとユーザ・ランドの二つから構成されています．狭義にLinuxと呼べるのは本来ならカーネルだけなのですが，Linuxカーネル上で動くように設計されたライブラリやユーティリティ・コマンドの類も広義の意味でLinuxシステムを構成する一部として考えても良いかもしれません．OCERAには独自拡張されたカーネルと，mini-rootを構成するためのユーザ・ランドの双方が含まれています．

OCERAを構成するコンポーネント

OCERAを構成する要素(コンポーネント)を以下に列挙します．

- コミュニケーション
- CANBUS
- ORTE(Ocera RealTime Ethernet)
- フォールト・トレラント(耐障害性)
- クオリティ・オブ・サービス(QoS)
- RTLinuxコンポーネント
- スケジューリング・コンポーネント
- POSIXインターフェース
- UDP stack

表1　PC向けLinuxと組み込みLinuxの特徴

PC向けLinux	組み込みLinux
高速Intel互換CPU	多様なRISC CPU
メモリ 128Mバイト以上	メモリ 128Mバイト以下
HDDあり	HDDなし(ストレージ容量4～64Mバイト)
汎用を考慮したプラットホーム	固定機能
ユーザがアプリケーションを追加	アプリケーション・ソフトを組み込んで出荷

表2　一般的な組み込みLinuxでの必要リソース

CPU	32ビット RISC CPU 数十～200MHz程度
メモリ	8～128Mバイト
ストレージ	フラッシュ・メモリ 4～128Mバイト

表3　RTLinux の機能拡張

● POSIX API ● POSIX Standard IO ● POSIX Priority Protection ● POSIX barriers ● POSIX Signals ● Handling of processer exception through POSIX signals ● POSIX Timers ● POSIX Message Queues ● POSIX Trace ● メモリ・マネージメント ● 動的メモリ管理（malloc and free） ● Big physical memory ● 共有メモリ（RTLinux スレッドと Linux プロセス間の） ● クロック・タイマ ● 同期時計 ● RTLinux 高解像度タイマ ● プロセス間通信 ● RTLinux スレッドと Linux プロセス間のウェイト・キュー ● RTLinux FIFO

表4　奨励されている OCERA 開発プラットホーム

ディストリビューション	Debian GNU/Linux
バージョン	3.0（woody）
カーネル・バージョン	2.4
ほかのツール類	docbook 4.2
	xpdf 1.00-3

表5　PCM5820 の仕様

メーカ	アドバンテック社
CPU	Geode 233MHz
メモリ	SO-DIMM 128M バイト
インターフェース	IDE，COM，Ethernet，Keyboard/Mouse，USB，VGA，FDD，PRINTER
電源	5V 単一電源　1～1.3A

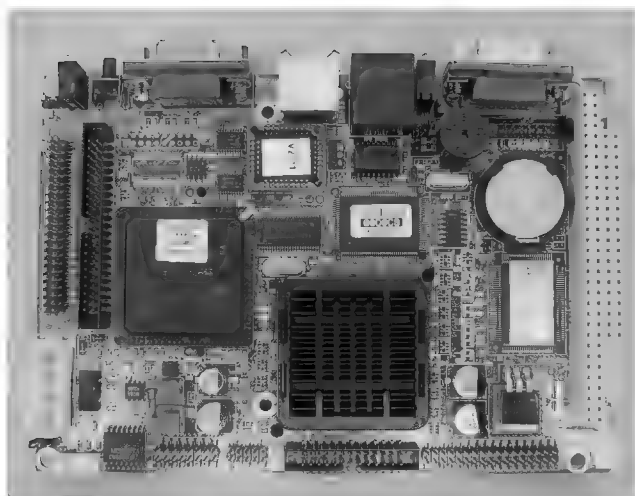


写真1　PCM5820（アドバンテック社）

● IDE disk interface

● Ada runtime

● Stand Alone RTLinux

開発中のものも含め，豊富なコンポーネントが用意されています．カーネルやデバイス・ドライバだけではなく，ユーザ・ランドの mini-root まで用意されています．mini-root に関して，どのようなコマンドを組み込むかのコンフィグレータも用意されています．現状では busybox 注1 をベースとしたミニマムな stand alone システムを構築することができるようです．

カーネルの機能

ocera-1.0.0には，以下に示すパッチを組み込み済みの linux-2.4.18カーネルが含まれています．以下のパッチはインターネットから探し出して，自分で単体でパッチを当てていくことも可能ですが，違う開発チームが開発しているパッチになるのでコンフリクトする恐れがあります．

しかし，ocera-1.0.0に含まれるカーネルにはすでにパッチと動作確認が取れているソース・コードなのでパッチ済みカーネルのみを利用しても十分に価値があると思われます．

● bigphys area patch
大容量メモリを使用するためのパッチ

● Low latency patch
カーネルやドライバ中にタスク切り替え可能箇所を見つけ，タスク切り替え処理を追加するパッチ

● Preempt io patch
カーネル空間実行中に割り込みが発生したときもタスク切り替えを行う

また，RTLinux に関しても表3に示す拡張が行われています．

開発/ターゲット・マシン

● 開発用マシン

OCERA プロジェクトのドキュメントにおいて，奨励されている開発環境は表4のとおりです．

Debian のパッケージ管理ツールである dpkg は使用しないのでほかのディストリビューションでもかまわないと思いますが，筆者は開発者と同じく Debian GNU/Linux 3.0（woody）を用意しました．

● ターゲット実行マシン

OCERA を実行するための PC を開発機とは別に用意しました．普通の PC でもかまいませんが，筆者が用意したものを写真1，表5に示します．

注1：busybox という一つのファイルの中に ls や cp，vi といった UNIX の基本コマンドを凝縮させた，小規模システムのためのユーティリティ．

インストールとコンフィグレーション，ビルド

それではさっそくOCERAをダウンロードしてみましょう．

`http://sourceforge.net/projects/ocera`

から`ocera-1.0.0.tar.gz`をダウロードできます．容量は比較的大きく，92Mバイトもあります．

ダウンロードしたら，開発用マシンでファイルを展開します．

```
$ tar xzfv ocera-1.0.0.tar.gz
```

`ocera-1.0.0.tar.gz`の構成ファイルは，図1のようになっています．

● コンフィグレーション

OCERAを展開しだディレクトリ`ocera-1.0.0/`の位置で，

```
$ make menuconfig
```

もしくは，

```
$ make xconfig
```

と入力するとOCERAコンフィグレーション画面が表示されます．普段使用しているカーネルのコンフィグレーション画面とほどんと同じです（図2）注2．

先にOSタイプを選択します．OCERAのドキュメントによれば，

- 10ms周期が必要ならソフト・リアルタイム
- 10μs周期が必要ならハード・リアルタイム（RTLinux+Linux）

としています．Linuxのtick（時間分解能）は10msなので，10ms以下の周期が必要なときにはハード・リアルタイムを選択することになります．

ただ，筆者の実験によれば，10μs周期はさすがにオーバ・ヘッドが大きくなりすぎました．実用的には50μs程度までと考えられます．

最初の実験ではハード・リアルタイムを選択します．

この画面では，

- Quality Of Service components
- Fault Tolerance components
- Communications components
  - CAN
  - ORTE（ocera realtime ethernet）

の取捨が選択できます．今回の実験ではこれら拡張機能は使わないこととします．画面下の，

RTLinux: Hard real-time --->

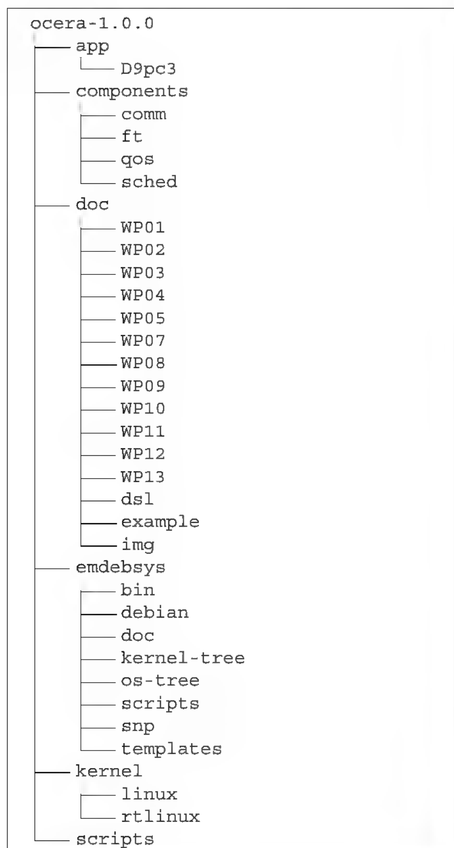
Linux Kernel Configuration --->

からLinux，RTLinuxカーネルの設定を行います．最後に保存して終了し，コンフィグレーションを完了します．

各コンポーネントは必要に応じて取捨すればよいのですが，注意事項としては，

- local APICは使わないほうが良い
- Virtual Terminal supportとVGA text consoleは使用する

という点に注意が必要かと思われます．



```
ocera-1.0.0
├── app
│   └── D9pc3
├── components
│   ├── comm
│   ├── ft
│   ├── qos
│   └── sched
├── doc
│   ├── WP01
│   ├── WP02
│   ├── WP03
│   ├── WP04
│   ├── WP05
│   ├── WP07
│   ├── WP08
│   ├── WP09
│   ├── WP10
│   ├── WP11
│   ├── WP12
│   ├── WP13
│   ├── dsl
│   ├── example
│   └── img
├── emdebsys
│   ├── bin
│   ├── debian
│   ├── doc
│   ├── kernel-tree
│   ├── os-tree
│   ├── scripts
│   ├── snp
│   └── templates
├── kernel
│   ├── linux
│   └── rtlinux
└── scripts
```

図1 OCERA1.0.0のファイル構成

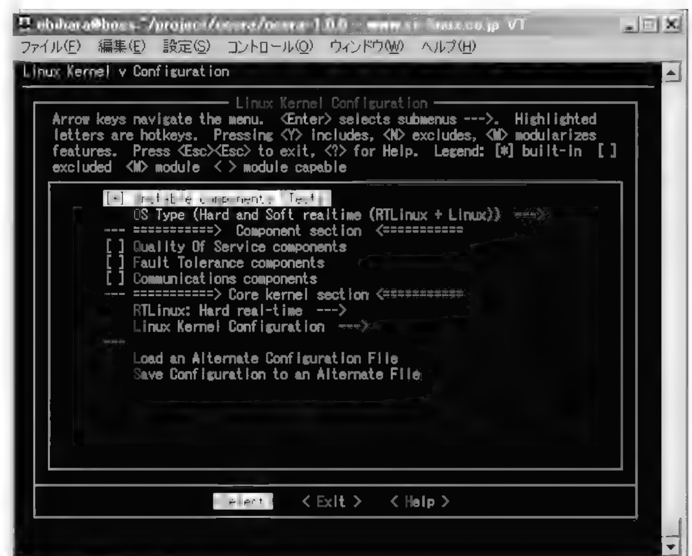


図2 OCERAのコンフィグレーション

注2：Xのコンフィグレータ使用時はQtライブラリを使用するため，事前に，

`apt-get install libqt3-dev`

として準備しておく必要がある．

● カーネルのビルド

ocera-1.0.0/のディレクトリ位置で，

```
$ make
```

を実行するだけで，kernel/linuxディレクトリで，

```
make dep bzImage modules
```

を実行したのと同じことをしてくれます．すなわち，カーネル(bzImage)とモジュールのコンパイル，RTLinuxモジュールのコンパイルまで行ってくれます．

ただし，残念なことにocera-1.0.0では，drivers/char/defkeymap.cがなく，bzImageリンク時にエラーが発生したので，筆者は素のlinux-2.4.18のdrivers/char/defkeymap.cをコピーして回避しました．

● target-i386ディレクトリへのインストール

target-i386ディレクトリが仮想的な組み込み装置のルート・ディレクトリになります．

```
$ su
```

図3
target-i386-tree

```
target-i386/boot/
  ├── System.map-2.4.18-ocera-1.0.0
  └── vmlinuz-2.4.18-ocera-1.0.0
target-i386/lib/modules/
 └── 2.4.18-ocera-1.0.0/
      ├── build -> /home/ebihara/project/ocera/ocera-1.0.0/kernel/linux/
      │   kernel/
      │    ├── drivers/
      │    │    ├── block/
      │    │    │    ├── loop.o
      │    │    │    └── rd.o
      │    │    ├── net/
      │    │    │    ├── bsd_comp.o
      │    │    │    ├── dummy.o
      │    │    │    ├── ppp_async.o
      │    │    │    ├── ppp_deflate.o
      │    │    │    ├── ppp_generic.o
      │    │    │    ├── ppp_synctty.o
      │    │    │    ├── pppoe.o
      │    │    │    ├── pppox.o
      │    │    │    ├── slhc.o
      │    │    │    └── wireless/
      │    │    │         ├── hermes.o
      │    │    │         ├── orinoco.o
      │    │    │         └── orinoco_plx.o
      │    │    ├── parport/
      │    │    │    ├── parport.o
      │    │    │    └── parport_pc.o
      │    │    └── usb/
      │    │         └── usbcore.o
      │    └── fs/
      │         ├── coda/
      │         │    └── coda.o
      │         ├── fat/
      │         │    └── fat.o
      │         ├── msdos/
      │         │    └── msdos.o
      │         ├── smbfs/
      │         │    └── smbfs.o
      │         └── vfat/
      │              └── vfat.o
      ├── lib/
      ├── misc/
      │    ├── mbuff.o -> ../../../../usr/rtlinux-ocera-1.0.0/modules/mbuff.o
      │    ├── rtl.o -> ../../../../usr/rtlinux-ocera-1.0.0/modules/rtl.o
      │    ├── rtl_fifo.o -> ../../../../usr/rtlinux-ocera-1.0.0/modules/rtl_fifo.o
      │    ├── rtl_ktrace.o -> ../../../../usr/rtlinux-ocera-1.0.0/modules/rtl_ktrace.o
      │    ├── rtl_malloc.o -> ../../../../usr/rtlinux-ocera-1.0.0/modules/rtl_malloc.o
      │    ├── rtl_posixio.o -> ../../../../usr/rtlinux-ocera-1.0.0/modules/rtl_posixio.o
      │    ├── rtl_sched.o -> ../../../../usr/rtlinux-ocera-1.0.0/modules/rtl_sched.o
      │    └── rtl_time.o -> ../../../../usr/rtlinux-ocera-1.0.0/modules/rtl_time.o
      ├── modules.dep
      ├── modules.generic_string
      ├── modules.ieee1394map
      ├── modules.isapnpmap
      ├── modules.parportmap
      ├── modules.pcimap
      ├── modules.pnpbiosmap
      ├── modules.usbmap
      └── pcmcia/
```

図4　起動画面

```
LILO 22.2 boot: rtl
Loading rtl.................
Linux version 2.4.18-ocera-1.0.0 (root@boss)
(gcc version 3.0.4) #16 2004年 7月 22日 木曜日 16:45:46 JST
BIOS-provided physical RAM map:
 BIOS-e820: 0000000000000000 - 00000000000a0000 (usable)
 BIOS-e820: 00000000000f0000 - 0000000000100000 (reserved)
 BIOS-e820: 0000000000100000 - 0000000007c00000 (usable)
 BIOS-e820: 00000000ffff0000 - 0000000100000000 (reserved)
On node 0 totalpages: 31744
zone(0): 4096 pages.
zone(1): 27648 pages.
zone(2): 0 pages.
Kernel command line: BOOT_IMAGE=rtl ro root=1601
                                    console=ttyS0,115200
Initializing CPU#0
Working around Cyrix MediaGX virtual DMA bugs.
Console: colour dummy device 80x25
Calibrating delay loop... 77.41 BogoMIPS
Memory: 122624k/126976k available (1198k kernel code,
         3968k reserved, 307k data, 232k init, 0k highmem)
Checking if this processor honours the WP bit even
                                 in supervisor mode... Ok.
Dentry-cache hash table entries:
                           16384 (order: 5, 131072 bytes)
Inode-cache hash table entries:
                            8192 (order: 4, 65536 bytes)
Mount-cache hash table entries:
                            2048 (order: 2, 16384 bytes)
Buffer-cache hash table entries:
                            4096 (order: 2, 16384 bytes)
Page-cache hash table entries:
                           32768 (order: 5, 131072 bytes)
Working around Cyrix MediaGX virtual DMA bugs.
CPU: Cyrix Geode(TM) Integrated Processor
                             by National Semi stepping 02
Checking 'hlt' instruction... OK.
POSIX conformance testing by UNIFIX
mtrr: v1.40 (20010327) Richard Gooch
                                   (rgooch@atnf.csiro.au)
mtrr: detected mtrr type: none
PCI: PCI BIOS revision 2.10 entry at 0xfada0, last bus=0
PCI: Using configuration type 1
PCI: Probing PCI hardware
PCI: Using IRQ router NatSemi [1078/0100] at 00:12.0
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
Starting kswapd
vesafb: framebuffer at 0x40800000, mapped to 0xc8800000,
                                               size 4032k
vesafb: mode is 1024x768x16, linelength=2048, pages=1
vesafb: protected mode interface info at c000:6df2
vesafb: scrolling: redraw
vesafb: directcolor: size=0:5:6:5, shift=0:11:5:0
Console: switching to colour frame buffer device 128x48
fb0: VESA VGA frame buffer device
vga16fb: mapped to 0xc00a0000
fb1: VGA16 VGA frame buffer device
Detected PS/2 Mouse Port.
pty: 256 Unix98 ptys configured
keyboard: Timeout - AT keyboard not present?(ed)
keyboard: Timeout - AT keyboard not present?(f4)
Serial driver version 5.05c (2001-07-08)
             with MANY_PORTS SHARE_IRQ SERIAL_PCI enabled
ttyS00 at 0x03f8 (irq = 4) is a 16550A
ttyS01 at 0x02f8 (irq = 3) is a 16550A
block: 128 slots per queue, batch=32
Uniform Multi-Platform E-IDE driver Revision: 6.31
ide: Assuming 33MHz system bus speed for PIO modes;
                               override with idebus=xx
CS5530: IDE controller on PCI bus 00 dev 92
CS5530: chipset revision 0
CS5530: not 100% native mode: will probe irqs later
    ide0: BM-DMA at 0xf000-0xf007,
                              BIOS settings: hda:pio, hdb:pio
    ide1: BM-DMA at 0xf008-0xf00f,
                              BIOS settings: hdc:pio, hdd:pio
hda: IRQ probe failed (0xfffffff8)
hda: IRQ probe failed (0xfffffff8)
hdb: IRQ probe failed (0xfffffff8)
hdb: IRQ probe failed (0xfffffff8)
hdc: CFA, ATA DISK drive
ide1 at 0x170-0x177,0x376 on irq 15
hdc: 64000 sectors (33 MB) w/4KiB Cache, CHS=500/4/32
Partition check:
 hdc: hdc1
FDC 0 is a post-1991 82077
Linux agpgart interface v0.99 (c) Jeff Hartmann
agpgart: Maximum main memory to use for agp memory: 91M
[drm] Initialized tdfx 1.0.0 20010216 on minor 0
[drm] Initialized radeon 1.1.1 20010405 on minor 1
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 8192 bind 16384)
ip_conntrack (992 buckets, 7936 max)
ip_tables: (C) 2000-2002 Netfilter core team
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
 hdc: hdc1
 hdc: hdc1
VFS: Mounted root (ext2 filesystem) readonly.
Freeing unused kernel memory: 232k freed
INIT: version 2.84 booting
Loading /etc/console/boottime.kmap.gz
Activating swap.
Checking root file system...
fsck 1.27 (8-Mar-2002)
/dev/hdc1: clean, 6998/8000 files, 26751/31984 blocks
/etc/init.d/rcS: System clock was not updated
                                            at this time.
Loading modules: af_packet modprobe: Can't locate module
                                                af_packet
musbio modprobe: Can't locate module musbio

Checking all file systems...
fsck 1.27 (8-Mar-2002)
Setting kernel variables.
Loading the saved-state of the serial devices...
/dev/ttyS0 at 0x03f8 (irq = 4) is a 16550A
/dev/ttyS1 at 0x02f8 (irq = 3) is a 16550A
Mounting local filesystems...
tmpfs on /tmp type tmpfs (rw)
tmpfs on /var type tmpfs (rw)
Cleaning: /etc/network/ifstate.
Setting up IP spoofing protection: rp_filter.
Configuring network interfaces: SIOCSIFADDR: No such
                                                   device
eth0: ERROR while getting interface flags: No such device
SIOCSIFNETMASK: No such device
eth0: ERROR while getting interface flags: No such device
done.
Starting portmap daemon: portmap.

Setting the System Clock using the Hardware Clock as
                                             reference...
modprobe: modprobe: Can't locate module char-major-10-135
System Clock set. Local time: Thu Jul 22 17:47:16 UTC 2004

Cleaning: /tmp /var/lock /var/run.
Recovering nvi editor sessions... done.
INIT: Entering runlevel: 2
Starting system log daemon: syslogd.
Starting kernel log daemon: klogd.
Starting internet superserver: inetd.
Starting deferred execution scheduler: atd.
Starting periodic command scheduler: cron.

SiliconLinux from Debian GNU/Linux 3.0 silinux ttyS0
```

```
# make linux
# make rtlinux
# make install_linux
# make install_rtlinux
```

この状態で，target-i386ディレクトリにターゲットに必要なファイル構成がコピーされています(図3，p.160)．

それでは，コンパイルされたカーネルtarget-i386/boot/vmlinuz-2.4.18-ocera-1.0.0でブートしてみます．

ブート・ローダにカーネルを認識させて起動する手順については，書籍やネット上に解説記事がたくさんあるので省略します．OCERAカーネルでの起動画面を図4(p.161)に示します．

ハード・リアルタイム性能の実験

それではOCERAカーネルのハード・リアルタイム性能の実験を行ってみます．OCERAカーネルは基本的にはRTLinuxのlinux-2.4.18対応と考えてさしつかえありません．そのため，ハード・リアルタイムの使用方法はRTLinuxと同じになります．プログラム"rt_thread.c"(リスト1)に示すようにリアルタイム・スレッドの周期を20μsとし，プリンタ・ポートのデータ・ビットに"H"/"L"を交互に繰り返してみます．パルスの周期は40μsになるので，25kHzの矩形波が出力されているようすが観察されます(図5)．linux-2.4.18でRTLinuxのリアルタイム・スレッドが動作するということになります．

リスト1　rt_thread.c

```
#include <rtl.h>
#include <time.h>
#include <pthread.h>
#include <asm/io.h>

#define LPT  0x378

pthread_t thread;

void init_hw(void){
  outb(0xc0,LPT+2);
}

void * start_routine(void *arg)
{
  int pulse=1;
  struct sched_param p;
  p . sched_priority = 1;
  pthread_setschedparam (pthread_self(), SCHED_FIFO, &p);

  /* 20nsec periodic */
  pthread_make_periodic_np (pthread_self(), gethrtime(), 20000);

  while (1) {
    outb(pulse,LPT);
    pulse=1-pulse;

    pthread_wait_np ();
  }
  return 0;
}

int init_module(void) {
  init_hw();
  return pthread_create (&thread, NULL, start_routine, 0);
}

void cleanup_module(void) {
  pthread_delete_np (thread);
}
```

OCERAのライセンス

さて，読者の皆様がもっとも関心を寄せる項目がライセンスに関してではないでしょうか．

本節ではドキュメントOCERA WhitePaperに基づいてライセンスに関して解説します．OCERAには，(全部ではないが)それ以前のオープン・ソース・プロジェクト由来のコードが含まれています．そのため，OCERAのライセンスを理解するためにはRTLinuxのライセンスを理解しなければなりません．

RTLinuxはFSMLabs社が開発を行っている商用ソフトという位置づけになります．RTLinuxのライセンスにはRTLinux/ProとRTLinux/Freeの2種類のライセンスが発行されています．RTLinux/Proはコマーシャル・ユースのための一般的な商用ソフトにおけるソフトウェア・ライセンスと同様な考え方と置くことができ，またRTLinux/Freeはオープン・ソースのためのライセンスになっています．

RTLinux/Freeを無償で使用するための条件は，

1) ユーザが作成するソフトウェアをGPLライセンスとする
もしくは，
2) OpenRTLinuxを改変せず，そのまま使用する場合はユーザ・プログラムをGPLにしなくても良い

すなわち，ユーザが作成するソフトをオープン・ソースにするつもりであればライセンス的にはまったく問題ありません．FSMLabs社が公開しているOpenRTLinuxをまったく改変せず，そのまま利用する場合にはユーザ・プログラムはクローズ・ソースでかまわないとされています．

ところが，たとえばOpenRTLinuxを最新のカーネルに対応



図5　リアルタイム・スレッド周期

させるなど，改変を行った場合にはユーザ・ソフトをクローズ・ソースにすることができません．OCERA ではRTLinux をlinux-2.4.18カーネルに対応させ，かつ独自の機能拡張を行っているので，OCERA で提供されているRTLinux 上で動作するユーザ・プログラム（カーネル空間，リアルタイム・スレッド）はGPL ライセンスを選択するか，FSMLabs 社からライセンスを購入することになります．

また，RTLinux にはその基本技術に関してアメリカ合衆国特許が成立しています．そのため，同じ基礎技術に基づく互換ソフトウェアの開発を独自に行うことも基本的には許されません．オープン・ソース技術に基づくソフトウェアに対しての特許の是非はここで議題しませんが，FSMLabs 社とFree Software Foundation は2001年10月に『GPL でカバーされるフリーソフトに関してはUSPatent No.5995745の使用料を支払わなくても良い』という合意に至ったとされています．

この観点からも，OCERA 上で動作するユーザ・プログラムはGPL ライセンスにしなければならないと考えられます．

さて，ここで述べている『ユーザ・プログラム』の範囲ですが，ここでいうユーザ・プログラムとは，RTLinux とリンクして実行される「カーネル空間，リアルタイム・スレッド」になります．Linux 上に“ロード”して動作する，プロセス空間プログラムのことではありません（図6）．Linux 上に通常と同じくロードして実行されるプロセスまではライセンス・ポリシがおよばないので独自ライセンスでかまいません．

そのため，組み込み装置にOCERA を用いる場合，ユーザが開発するソフトウェアのライセンスは，以下のようになると考えられます．

1) カーネル空間リアルタイム・スレッド部分のユーザ開発プログラムはGPL にしなければならない
2) Linux 上にロードされるプロセス部分のユーザ開発プログラムは独自ライセンスでかまわない

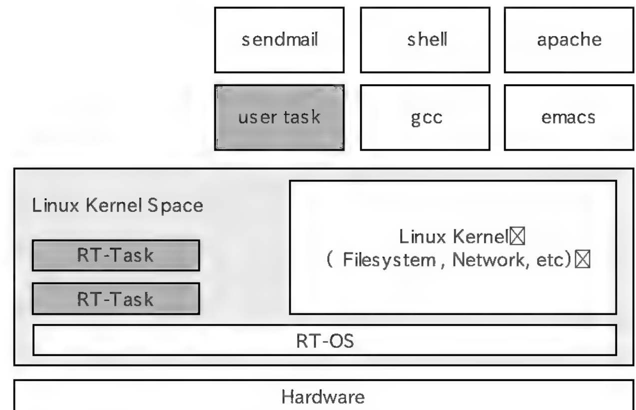


図6　RTLinux の構造

どちらの部分のプログラムも著作権はプログラム作者自身になります．プログラムを開発し，自分自身が著作者となり，GPL ライセンス，もしくは独自ライセンスの元にリリースする形式となります．WhitePaper によれば，GPL の回避が必要であればRTLinux/Proのライセンスの取得が必要であろうとしています．

おわりに

以上，OCERA プロジェクトの紹介をしました．今回は動作確認までしか至らず，OCERA の各コンポーネントの評価まではできませんでしたが，今後期待が高まるプロジェクトであることにはまちがいありません．ソフト・リアルタイムやQoS などの各コンポーネントも評価・実験を行ってみたいと考えます．

えびはら・ゆうたろう　シリコンリナックス（株）

x86CPUだけでもマスタしたい

開発技術者のためのアセンブラ入門

第㉙回［最終回］ アセンブラを使いこなすための基礎知識と C言語からのアセンブラの使用方法（gas編：その2）

大貫 広幸

今回は，Linuxで使われているC/C++言語のコンパイラGCCのインライン・アセンブラの機能と使い方について説明します．

GCCのインライン・アセンブラの構文

GCCでインライン・アセンブラを使用する場合，キーワード「asm」を使います．

C/C++言語のソース・プログラム上において，asmは，

```
asm(…);
```

あるいは

```
asm volatile(…);
```

のようにパラメータを指定するかっこ()と文の終わりを示すセミコロン(;)を付ける形で記述します．

ここでasmにおけるvolatileの役割を説明します．volatileの指定は，asmをコンパイラの最適化の対象とするか否かに関係します．volatileがない場合，asmは最適化の対象となります．最適化の対象となったasmは，最適化処理の過程で移動されたり，削除される可能性があります．

しかし，volatileがあるasmは最適化の対象とはならず，最適化処理の過程で移動されたり，削除される可能がなくなります．

asmのかっこ内のパラメータは，左から「アセンブリ命令」，「出力オペランド」，「入力オペランド」，「変更レジスタ」の順に四つのフィールドが記述され，各フィールドはコロン(:)によって区切られます(図1)．

出力オペランド，入力オペランド，変更レジスタの各フィールドには，0個以上の項目が記述されます．一つのフィールドに複数の項目を記述する場合は，カンマ(,)によって項目を区切ります．

項目がない，つまり使用しないフィールドは，0個以上の空白を記述しておきます．ただし，右端から連続して使用されないフィールドがある場合，その右端から連続して使用されないフィールドは，コロン(:)を含めて省略します．

たとえば，アセンブリ命令をA，出力オペランドをO，入力オペランドをI，変更レジスタをRで表した場合，すべてのフィールドが使われている場合は，

```
asm(A:O:I:R);
```

となります．

しかし，アセンブリ命令と変更レジスタのみ使用されている場合なら，

```
asm(A:::R);
```

と記述します．

また，入力オペランド，変更レジスタの指定がなければ入力オペランドから後のフィールドを完全に省略し，

```
asm(A:O);
```

と記述します．

● 出力オペランド，入力オペランド，変更レジスタの役割

インライン・アセンブラのプログラムは，asmのアセンブリ命令のフィールドに記述されます．そして，出力オペランド，入力オペランド，変更レジスタの各フィールドは，このアセンブリ命令のフィールドに記述されたアセンブラのプログラムと，C/C++言語側のプログラムをつなぐ役割をしています．

GCCの場合，コンパイラはアセンブリ命令のフィールドの内容を解析しません．基本的にはアセンブリ命令の記述フィールドの内容は，そのままgasのアセンブラに渡されることになります．

これは，GCCがいろいろな種類のCPUをサポートしていることからきています．CPUの種類が違えばアセンブリ命令も異なります．そのため，キーワードasm一つのために，GCC内にアセンブリ命令の解析ルーチンを組み込むのは，たいへんな作業となります．

```
asm[ volatile](アセンブリ命令[:出力オペランド][:入力オペランド]:変更レジスタ);⊠
asm[ volatile](アセンブリ命令[:出力オペランド]:入力オペランド);⊠
asm[ volatile](アセンブリ命令[:出力オペランド]);
```

図1 GCCのインライン・アセンブラasmの構文

(注)[ ]は省略可能な内容を示す．

そのため，GCCのasmはアセンブリ命令のフィールドの内容を解析しない代わりに，出力オペランド，入力オペランド，変更レジスタを使い，アセンブリ命令の内容をコンパイラに伝えています．

● 変更レジスタの記述フィールド

変更レジスタのフィールドは，アセンブリ命令のフィールドに書かれた機械語命令を実行した場合，内容が変化するレジスタをGCCのコンパイラに知らせるためのものです．

コンパイラはソース・プログラムをコンパイルし，機械語コードを生成するために，つねにCPUのレジスタの状態を把握している必要があります．

先に述べたようにGCCは，asmのアセンブリ命令のフィールドの内容を解析しません．そのため，アセンブリ命令で使われているレジスタが何か，GCCのコンパイラは知ることができません．

そこで，入出力オペランドと，この変更レジスタのフィールドで，アセンブリ命令で使われているレジスタを指定するわけです．変更レジスタのフィールドには，入出力オペランドでは指定されないアセンブラ命令のフィールドで直接指定されているレジスタ，この場合は内容が変更されるレジスタを指定するわけです．

GCCのコンパイラは，変更レジスタのフィールドの内容を見て，必要ならasmのインライン・アセンブラの機械語命令の実行の前後で，内容が変わるレジスタの値の退避と復元を行うコードを生成します．

逆のいいかたをすると，変更レジスタのフィールドに記述されていないレジスタの内容をアセンブラのプログラムで変更すると，レジスタ値変更による実行段階での思いもかけない不具合が発生する場合もあるので，この点は注意が必要です．

アセンブラのプログラムで変更されるレジスタは，gasで使われているレジスタ名を二重引用符(")で囲み指定します．たとえば，レジスタEAXが変更されるのなら，

```
"%eax"
```

と記述します．

86系の32ビットCPUのGCCの場合，1語を32ビットとしているため，汎用レジスタの退避と復元も32ビット単位で行われます．そのため，8ビットや16ビットのレジスタ名を使用しても32ビットでの退避と復元となります．

たとえば，レジスタEBXとDX，CLの三つのレジスタを変更するのなら，

```
"%ebx","%dx","%cl"
```

と記述するわけですが，実際の退避と復元はレジスタEBX，EDX，ECXとなります．

そのため，32ビット・レジスタ名を使って，

```
"%ebx","%edx","%ecx"
```

と記述することもできます．

● アセンブリ命令の記述フィールド

アセンブリ命令は二重引用符(")で囲み，C/C++言語の文字列定数として記述します．たとえば，NOP命令なら，

```
asm("nop");
```

と記述します．

ただし，通常のC/C++言語の文字列定数とは異なり，文字列定数内に物理的な改行コードを含めることができます．これによりGCCのasmは，複数行におよぶアセンブリ命令を，一つのasm文で記述することが可能になっています．

また，一つのasm文で複数行におよぶアセンブリ命令を記述する場合には，GASで使われているシャープ(#)によるコメントも記述することができます．

リスト1は，いろいろなアセンブリ命令の記述例を示したものです．

① アセンブリ命令のフィールド内でのC/C++言語側の変数の使用

前にも述べたように，アセンブリ命令のフィールドの内容は，そのままgasに渡されます．そのため，アセンブリ命令のフィールド内に記述された変数名などは，そのままの形でgasに渡されます．

GCCのコンパイラの場合，グローバルな変数は，定義された名前でgasに渡されるため，アセンブリ命令のフィールド内でもグローバルな変数は使用可能です．

しかし，関数内やブロック内で使われているローカルな変数や関数の仮引き数は，コンパイル後はその名前でgasには渡されないため，アセンブリ命令のフィールド内ではこのような変数は使用できません(リスト2)．

この不便な点を解決するのが，入出力オペランドのフィールドです．入出力オペランドを使用すると，関数内やブロック内で使われているローカルな変数や関数の仮引き数もアセンブリ命令のフィールド内で扱うことが可能となります．

② インライン・アセンブラ内でのラベルとジャンプ

アセンブリ命令のフィールド内では，gasと同じようにラベルを定義し，そこへジャンプすることができます．

しかし，①で述べたのと同じ理由から，関数内やブロック内で使われているC/C++言語のラベルは，コンパイル後，別の名前となりgasに渡されます．そのため，asm文のアセンブラ・プログラムからC/C++言語側のラベルへのジャンプや，その逆のC/C++言語側のgoto文でasm文のアセンブラ・プログラム内のラベルへのジャンプはできません(リスト3)．

③ 入出力オペランドを使用するための準備

アセンブラ命令のオペランドの部分でパーセント(%)の後に数字を書いたもの(%0,%1,%2,%3,…)を使用することで，アセンブラ命令のオペランドを任意の文字列に変更できます．この%0,%1,%2,%3,…は，asm文に出力オペランドおよび入力オペランドが記述されている場合に使用できます．

%0,%1,%2,%3,…は，コンパイル時にGCCが出力オペラン

リスト 1　GCCにおけるインライン・アセンブラ asm の記述例

```
#include <stddef.h>
#include <stdio.h>

unsigned long varA;

unsigned long IlAsm1(long cnt)
{
    unsigned long varB;
    if( cnt==0 ) return varA;
    if( cnt>0 ) {
      asm volatile("movl  %0,%%edx" ::"m"(varA):"%edx");
      asm volatile("movb  %0,%%cl"  ::"m"(cnt) :"%cl");
      asm volatile("roll  %%cl,%%edx" /* 左へCLビット回転 */ :::"%edx");
      asm volatile("movl  %%edx,%0" :"=m"(varB));
    }
    else { /* cnt<0 */
      asm volatile("movl  %0,%%edx" ::"m"(varA):"%edx");
      asm volatile("movb  %0,%%cl"  ::"m"(cnt) :"%cl");
      asm volatile("rorl  %%cl,%%edx" /* 右へCLビット回転 */ :::"%edx");
      asm volatile("movl  %%edx,%0" :"=m"(varB));
    }
    return varB;
}

unsigned long IlAsm2(long cnt)
{
    unsigned long varB;
    if( cnt==0 ) return varA;
    if( cnt>0 ) {
      asm volatile("movl %0,%%edx; movb %1,%%cl"::"m"(varA),"m"(cnt):"%edx","%cl");
      asm volatile("roll %%cl,%%edx /* 左へCLビット回転 */; movl %%edx,%0":"=m"(varB)::"%edx");
    }
    else { /* cnt<0 */
      asm volatile("movl %0,%%edx; movb %1,%%cl"::"m"(varA),"m"(cnt):"%edx","%cl");
      asm volatile("rorl %%cl,%%edx /* 右へCLビット回転 */; movl %%edx,%0":"=m"(varB)::"%edx");
    }
    return varB;
}

unsigned long IlAsm3(long cnt)
{
    unsigned long varB;
    if( cnt==0 ) return varA;
    if( cnt>0 ) {
      asm volatile("
        movl    %1,%%edx
        movb    %2,%%cl
        roll    %%cl,%%edx    # 左へCLビット回転
        movl    %%edx,%0
        ":"=m"(varB):"m"(varA),"m"(cnt):"%edx","%cl");
    }
    else { /* cnt<0 */
      asm volatile("
        movl    %1,%%edx
        movb    %2,%%cl
        rorl    %%cl,%%edx    # 右へCLビット回転
        movl    %%edx,%0
        ":"=m"(varB):"m"(varA),"m"(cnt):"%edx","%cl");
    }
    return varB;
}
```

```
        pushl %ebp
        movl %esp,%ebp
        subl $4,%esp
             ∫
#APP
        movl   varA,%edx
        movb   8(%ebp),%cl
        roll   %cl,%edx
        movl   %edx,-4(%ebp)
#NO_APP
             ∫
#APP
        movl   varA,%edx
        movb   8(%ebp),%cl
        rorl   %cl,%edx
        movl   %edx,-4(%ebp)
#NO_APP
             ∫
        movl -4(%ebp),%edx
        movl %edx,%eax
        leave
        ret
```



```
● 例1：出力オペランド1，入力オペランド0☒
  asm("…%0…":  ～  );

● 例2：出力オペランド0，入力オペランド1☒
  asm("…%0…"::  ～  );

● 例3：出力オペランド1，入力オペランド1☒
  asm("…%0…%1…":  ～  :  ～  );

● 例4：出力オペランド2，入力オペランド3☒
  asm("…%0…%1…%2…%3…%4…":  ～, ～  :  ～, ～, ～  );
```

（注）～は入出力オペランドの一つの項目を表わす☒

図 2　%0,%1,%2,%3,…と入出力オペランドの各項目の対応例

ドおよび入力オペランドの各項目を評価し，その結果を文字列として置き換えます．そして，gasには置き換え後の文字列が渡されます．

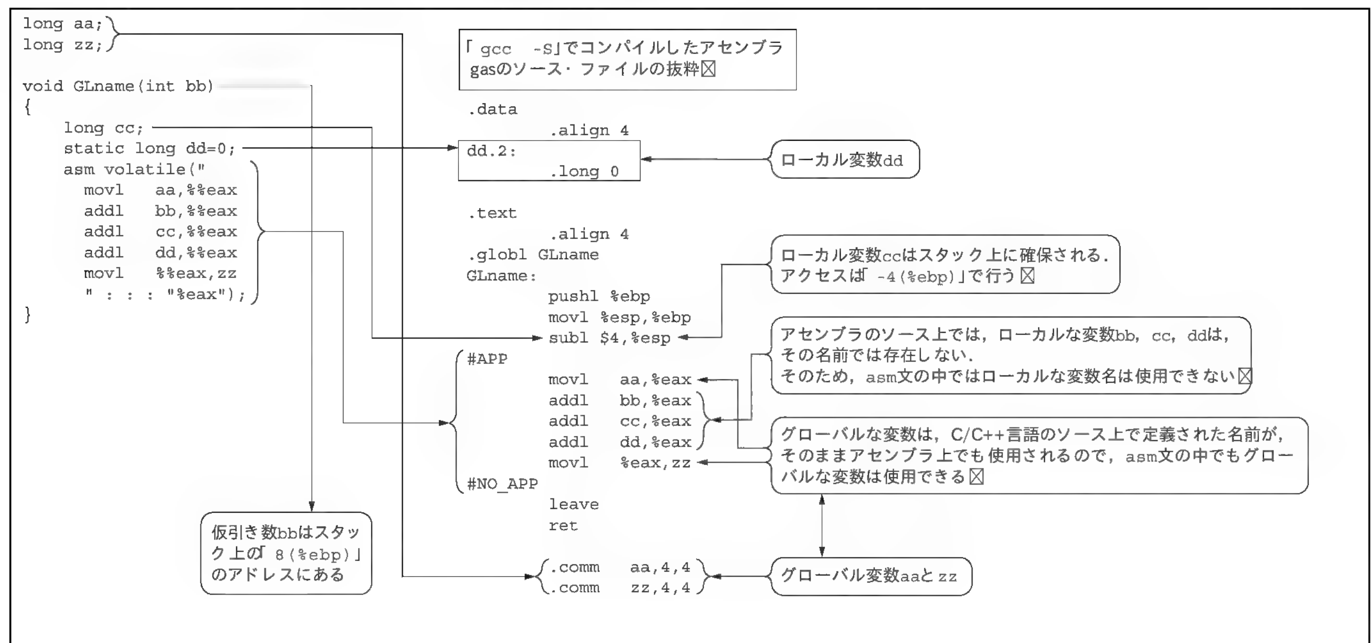
入出力オペランドと %0,%1,%2,%3,…の使用は，いわゆるマクロの動作と似ています．つまり，%0,%1,%2,%3,…が仮引き数，出力オペランドおよび入力オペランドの各項目が実引き数ということです．

%0,%1,%2,%3,…の対応は，出力オペランド，入力オペランドの順で連続させた状態で，各項目を記述上の左端を 0として右へ連番で対応しています（図 2）．

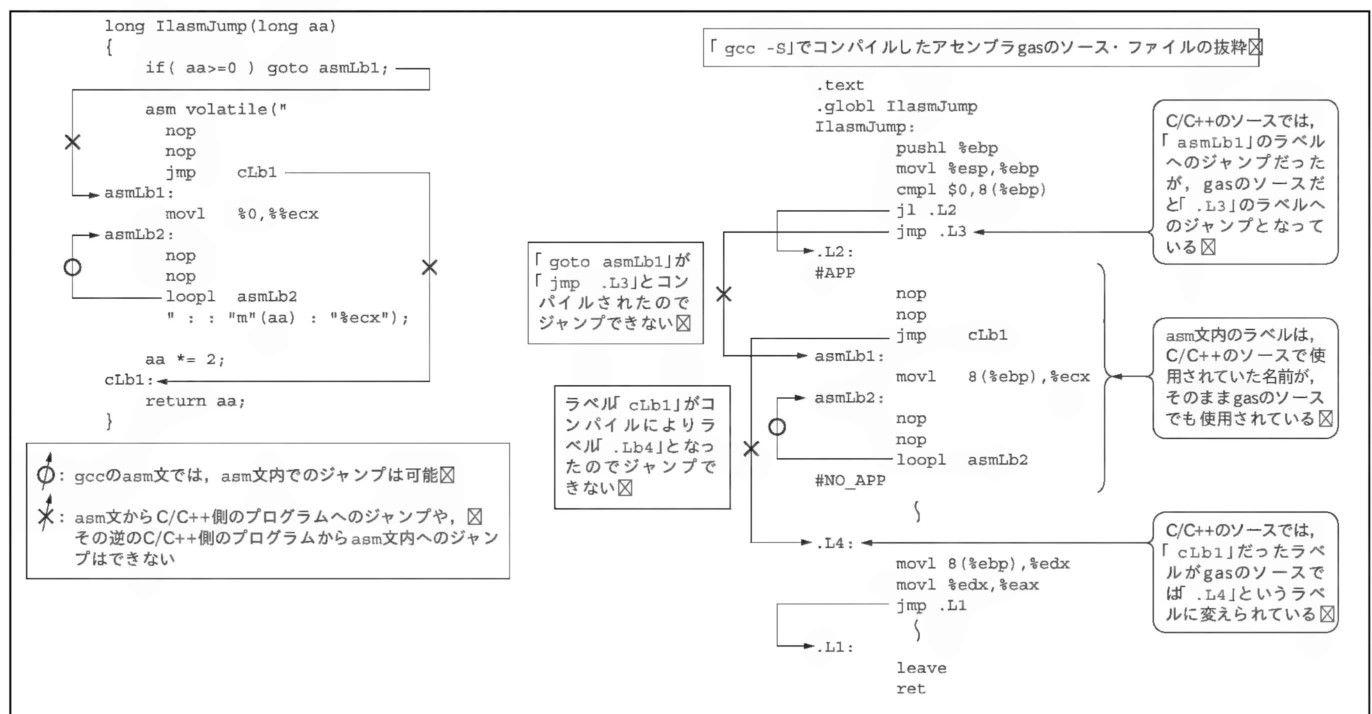
④ %0,%1,%2,%3,…を使用する場合の注意

ご存じのように 86系 CPU の gas では，レジスタ名には頭に %

リスト 2　インライン・アセンブラ内のグローバル変数とローカル変数



リスト 3　インライン・アセンブラ内のジャンプ



が付きます．また，%0,%1,%2,%3,…の仮引き数も頭に%が付きます．

そのため，レジスタ名と%0,%1,%2,%3,…の仮引き数を区別するため，入出力オペランドおよび変更レジスタの指定がある場合には，レジスタ名の頭に付く%は「%%」と二つにする必要があります．

● 入力オペランドの記述フィールド

入力オペランドは，C/C++言語側のプログラムの値を，アセンブリ命令のフィールドにあるアセンブラ側のプログラムに渡すためのものです．

入力オペランドの一つの項目は，アセンブラ側のオペランドの型を示す「Constraint」と，そのオペランドに入る「変数」あるいは値を示す「式」からなります．

Constraintは，二重引用符（"）で囲んで記述します．それに続きC/C++言語の変数あるいは式をかっこ()で囲んで記述します．

おもに使われるConstraintとしては表1のようなものがあります．ちなみに，Constraintの詳細はinfoにそのドキュメントがあります．infoでは，[gcc]→[Machine Desc]→[Constraints]と進むとConstraintのドキュメントにたどり着きます．

Constraintの中でも，メモリを表すmと32ビット・レジスタを表すr，そして32ビット・レジスタあるいはメモリを示すgあたりがもっとも使用されるConstraintといえます．

① m

mはメモリをアクセスするオペランドで使用します．この場合，受け側のアセンブリ命令のオペランド（%数字）はメモリである必要があります．しかし，送り側のC/C++言語の変数，あるいは式で使われる変数は，グローバル変数でもローカルな変数でも，関数の仮引き数でもかまいません．

たとえば，C/C++言語の変数valXそのものをアセンブラ側に渡すのであれば，

```
"m"(valX)
```

と記述します．

ここでは"m"(valX)が，アセンブリ命令のフィールドの%3に対応するとした場合，矢印の左辺をアセンブリ命令のフィールド，右辺をコンパイル後，gasに渡される文字列とした場合，

「movw　%3,%%ax」 →「movw　varX,%ax」

のようになります（リスト4，list4aの関数）．

かっこ内に式が指定されていた場合，GCCのコンパイラは，事前にかっこ内の式をコンパイルしてコードを生成します．そして式の結果（32ビット値）をコンパイラが確保したメモリにストアする機械語命令を自動生成します．アセンブリ命令のフィールドに対しては，式の結果が入った32ビットのメモリ（コンパイラが確保したメモリ）を受け側のアセンブリ命令のオペランドとします（リスト4，list4b，list4cの関数）．

② r

rは32ビット・レジスタをアクセスするオペランド（%数字）で使用します．

rの場合，GCCのコンパイラは事前にかっこ内の式をコンパイルしコードを生成します．そして，式の結果（32ビット値）をコンパイラが選んだ32ビット・レジスタにロードする機械語命令を自動生成します．アセンブリ命令のフィールドに対しては，式の結果が入った32ビット・レジスタを受け側のアセンブリ命令のオペランドとします．

この場合は，受け側のアセンブリ命令のオペランドは32ビット・レジスタである必要があります．ただし，使用されるレジスタはGCCのコンパイラが勝手に決めた32ビット・レジスタとなります．送り側の式で使われるC/C++言語の変数は，先のmのときと同じで，グローバル変数でもローカル変数でも，関数の仮引き数でもかまいません．

たとえば，C/C++言語の変数valXの32ビット値をアセンブラ側の32ビット・レジスタに渡すのであれば"r"(valX)とします．

この場合GCCのコンパイラは，事前にvalXの32ビットの値をメモリからコンパイラが選択した32ビット・レジスタにロードする機械語命令を自動生成します．そして，その32ビット・レジスタ（コンパイラが選択したレジスタ）を受け側のアセンブリ命令のオペランドとします（リスト5，list5aの関数）．

また，「a+b*c」のようなC/C++言語の式をアセンブラのレジスタに渡すのであれば，

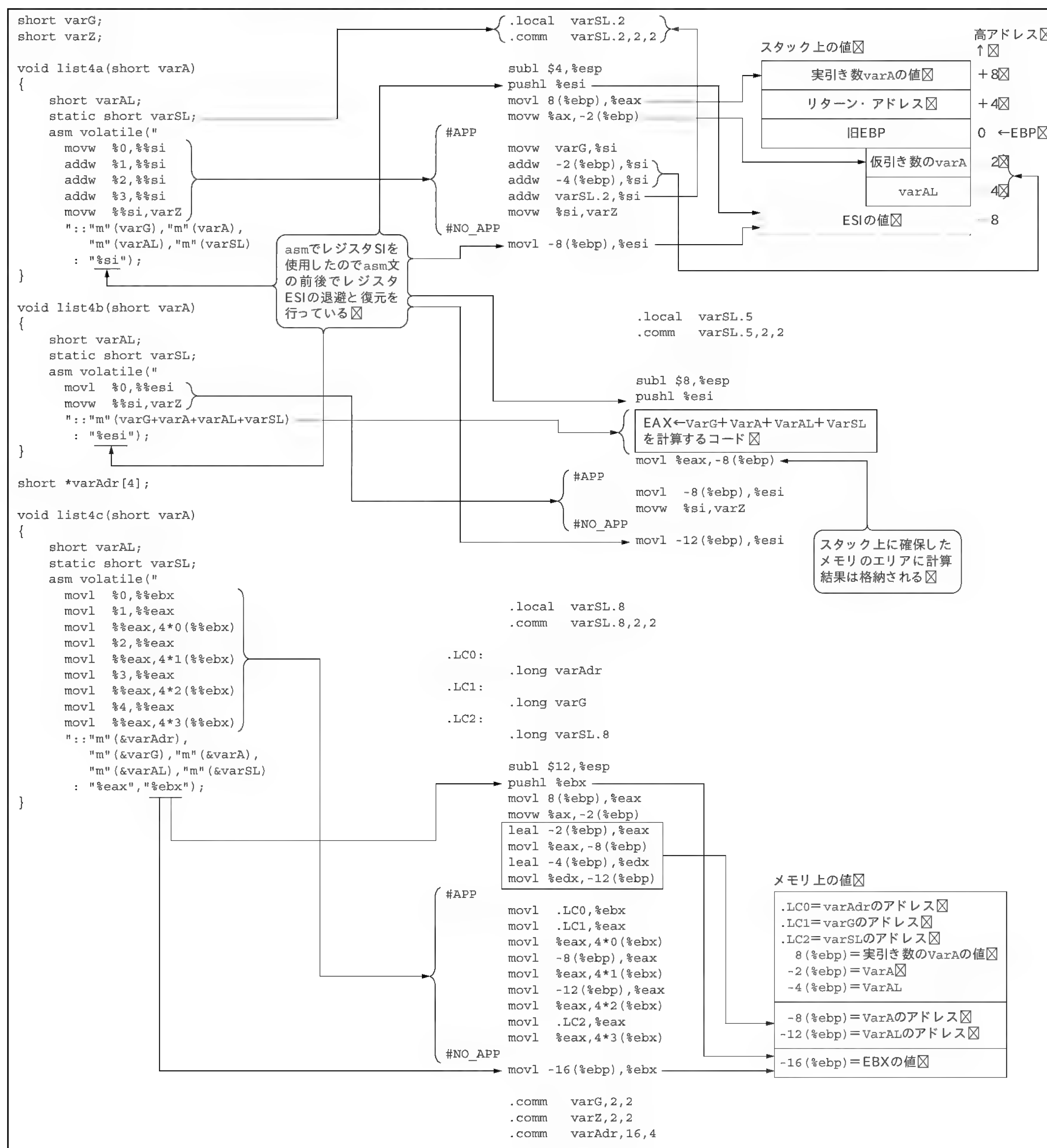
```
"r"(a+b*c)
```

と記述します．この場合，事前にa+b*cの演算を行うコードが生成され，演算結果をコンパイラが選択した32ビット・レジ

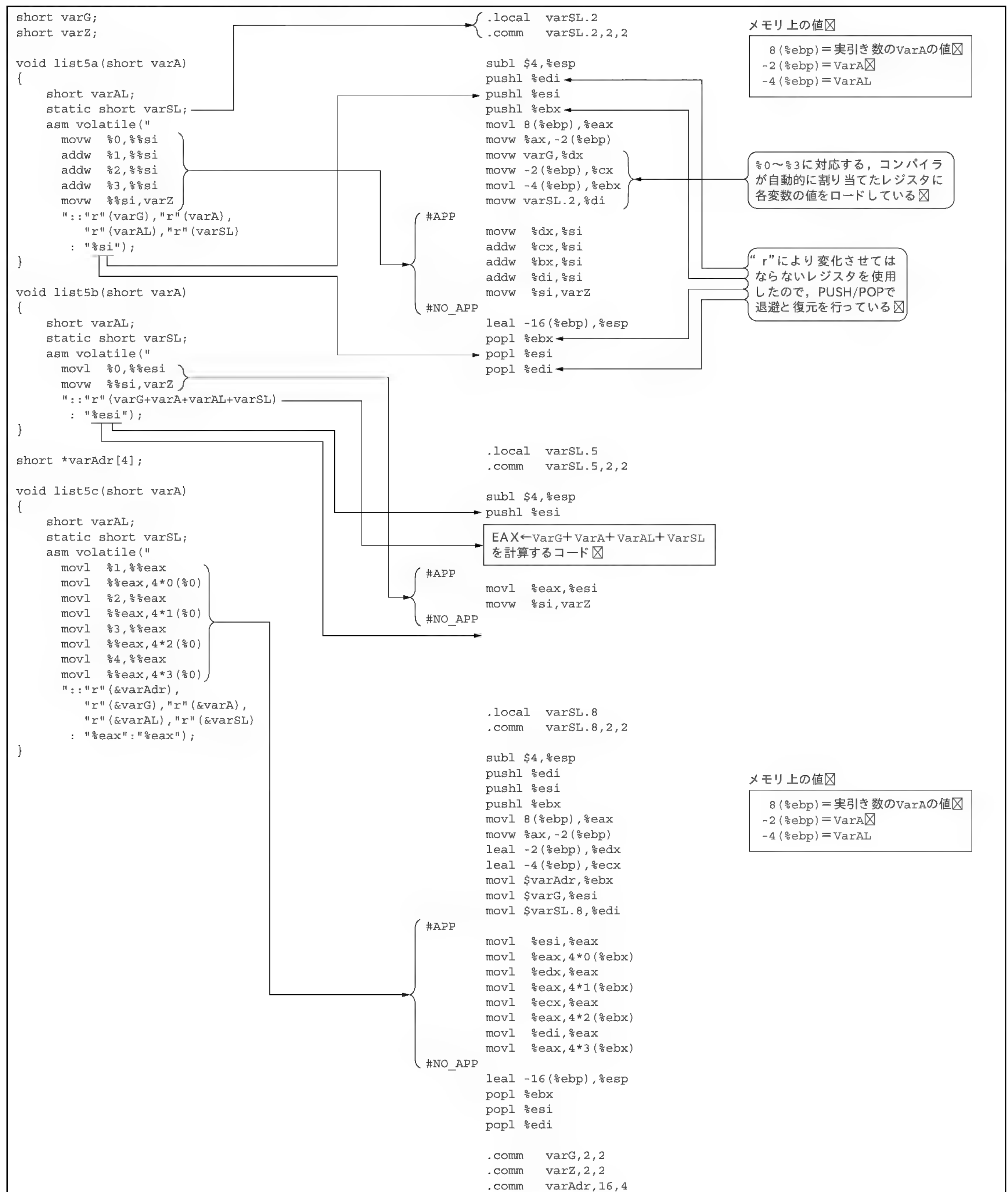
表1　おもなConstraint

Constraint	内　容
a	レジスタEAX
b	レジスタEBX
c	レジスタECX
d	レジスタEDX
S	レジスタESI
D	レジスタEDI
A	レジスタEDXとEAXを連結した64ビット長のレジスタ・ペア
q	レジスタEAX，EBX，ECX，EDXの内のいずれか一つのレジスタ
r	レジスタEAX，EBX，ECX，EDX，ESI，EDIの内のいずれか一つのレジスタ
t	FPUのレジスタ・スタックST(0)
u	FPUのレジスタ・スタックST(1)
f	FPUのレジスタ・スタックST(0)～ST(7)のうちのいずれか一つのレジスタ
m	メモリ
i　n	イミディエイト
g	汎用レジスタ/メモリ/イミディエイトのいずれか一つが選択される
0 1 2 … 9	「%数字」に対応する入出力オペランドに割り当て済みのレジスタやメモリ

リスト 4　入力オペランドにおける Constraint"m"の動作



リスト 5　入力オペランドにおける Constraint“r”の動作



スタに転送する機械語命令を自動生成します．そして，その32ビット・レジスタ(コンパイラが選択したレジスタ)を受け側のアセンブリ命令のオペランドとします(リスト5，list5bの関数)．

C/C++言語の式には，変数のアドレスを求める&の演算子も使用できます．たとえば，C/C++言語の変数valXのアドレス(32ビット長)をアセンブラ側の32ビット・レジスタに渡すのであれば，

"r"(&valX)

とします．

この場合，GCCのコンパイラは，事前にvalXのアドレス値をコンパイラが選択した32ビット・レジスタにロードする機械語命令を自動生成します．そして，その32ビット・レジスタ(コンパイラが選択したレジスタ)を受け側のアセンブリ命令のオペランドとします(リスト5，list5cの関数)．

rのように使用レジスタの選択をコンパイラに任せるのではなく，プログラマが決めたレジスタにするのであれば，表1のa～d，S，Dを使用します．

③g

gはレジスタあるいはメモリの両方が指定可能なオペランドで使用します．gを使用すると，汎用レジスタとメモリの中から最適なレジスタあるいはメモリをコンパイラが自動的に選択して使用します．

つまりgは，簡単にいうと，これまで説明したmとrを一つにしたものといえます．

● 出力オペランドの記述フィールド

出力オペランドは，アセンブリ命令のフィールドにあるアセンブラ側のプログラムの値を，C/C++言語側のプログラムに渡すためのものです．

出力オペランドの一つの項目は，出力の状態を表す「Modifier Characters」とアセンブラ側のオペランドの型を示す「Constraint」，そのオペランドの送り先となる「変数」からなります．

Modifier CharactersとConstraintは，二重引用符(")で囲んで記述します．それに続きC/C++言語の変数をかっこ()で囲み記述します．受け側のC/C++言語の変数は，グローバル変数でもローカルな変数でも，関数の仮引き数でもかまいません．

Modifier Charactersにもいくつもの種類があります．その詳細は，infoのConstraintと同じ場所に，そのドキュメントがあります．

しかし，出力オペランドで実際によく使われるModifier Charactersは，「書き込み専用」を表すイコール(=)ぐらいです．

Constraintは，入力パラメータで使用したものと同一のものです．出力オペランドの場合，アセンブラ側の値を直接C/C++言語側の変数に格納するのならm，任意のレジスタを経由して間接的にC/C++言語側の変数に格納するのならr，直接でもレジスタ経由のどちらでもかまわない場合にはgを使用します．

たとえば，C/C++言語の変数varZにアセンブラ側の値を格納するのであれば，mを使用する場合は"=m(varZ)"，rを使用するのなら"=r(varZ)"，そしてgを使用するのなら"=g(varZ)"と記述します(リスト6)．

mを使ってアセンブラ側の値を直接C/C++言語側の変数に格納する場合，アセンブリ命令の送り先のオペランドはメモリである必要があります．また，基本的にはアセンブラ側のメモリのデータ長は，送り先のC/C++言語側の変数のサイズと同じにします．しかし，意図的にメモリのデータ長を，送り先のC/C++言語側の変数のサイズと異なる長さにすることも可能です．

rのレジスタ経由でのC/C++言語側の変数への格納は，32ビット・レジスタが使用されます．そのため，アセンブリ命令の送り先のオペランドは32ビット・レジスタである必要があります．

また，同じ理由から，gを使用する場合も，アセンブリ命令の送り先のオペランドは32ビットのレジスタ，あるいは32ビット長のメモリである必要があります．

GCCのインライン・アセンブラの使用例

ここでは，実際のインライン・アセンブラの使用例を紹介しながら，もう少しGCCのインライン・アセンブラの使い方を説明します．

● MMXおよびSSE/SSE2命令の使用

リスト7はMMXおよびSSE/SSE2命令の使用例です．

MMXやSSE/SSE2命令は，アセンブラgasがサポートしていれば，GCCのインライン・アセンブラでも，MMXやSSE/SSE2命令を使用することができます．

MMXの場合，以前MMX命令を説明したときMMXのレジスタは，FPUのレジスタ・スタックと物理的に同じレジスタを共有していると述べました．そのため，インライン・アセンブラでMMX命令を使用する場合は，ST(0)～ST(7)のFPUのレジスタ・スタックの内容は破壊されてなくなることになります．そのため，変更レジスタでそのことをコンパイラに伝えることが必要になります．

● 数字のConstraintの使用例

86系に限らずCPUの二項演算の機械語命令は，転送先をDEST，転送元をSOU，そして演算をopで表すと，

DEST op SOU → DEST

と動作します．

つまりこの場合，DESTは転送先である前に転送元としても使われることになります．このような機械語命令をインライン・アセンブラで使用する場合，一つのオペランドが転送先と転送元の二つの属性をもつことになります．

通常，このような命令をインライン・アセンブラで記述する場合，一つのオペランドには転送先か転送元のどちらかしか指定できません．そのため，目当ての機械語命令は一つなのに，その一つの機械語に付随してメモリからのレジスタへの値の

リスト 6　出力オペランドの動作

```
short varG;
short varZ;

void list6a(short varA)
{
    short varAL;
    static short varSL;
    asm volatile("
      movw   %4,%%si
      movw   %%si,%0
      movw   %%si,%1
      movw   %%si,%2
      movw   %%si,%3
      ":"=m"(varG),"=m"(varA),
        "=m"(varAL),"=m"(varSL)
       :"m"(varZ):"%si");
}
```

```
        .local  varSL.2
        .comm   varSL.2,2,2

        subl $4,%esp
        pushl %esi
        movl 8(%ebp),%eax
        movw %ax,-2(%ebp)
#APP
        movw  varZ,%si
        movw  %si,varG
        movw  %si,-2(%ebp)
        movw  %si,-4(%ebp)
        movw  %si,varSL.2
#NO_APP
        movl -8(%ebp),%esi
```

メモリ上の値☒
8(%ebp)=実引き数のvarAの値☒
-2(%ebp)=VarA☒
-4(%ebp)=VarAL

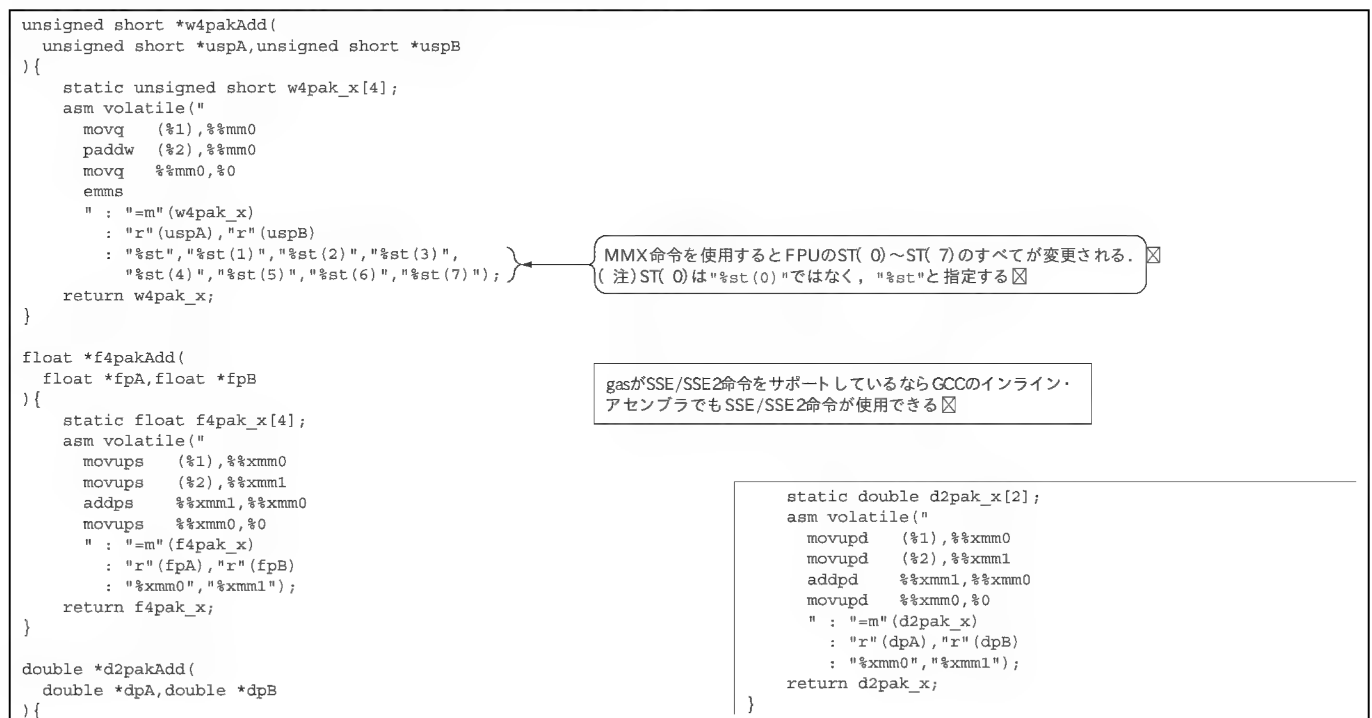
```
void list6b(short varA)
{
    short varAL;
    static short varSL;
    asm volatile("
      movw   %4,%%si
      movw   %%si,%0
      movw   %%si,%1
      movw   %%si,%2
      movw   %%si,%3
      ":"=r"(varG),"=r"(varA),
        "=r"(varAL),"=r"(varSL)
       :"m"(varZ):"%si");
}
```

```
        .local  varSL.5
        .comm   varSL.5,2,2

        subl $4,%esp
        pushl %edi
        pushl %esi
        pushl %ebx
        movl 8(%ebp),%eax
        movw %ax,-2(%ebp)
#APP
        movw  varZ,%si
        movw  %si,%dx
        movw  %si,%cx
        movw  %si,%bx
        movw  %si,%di
#NO_APP
        movw %dx,varG
        movw %cx,-2(%ebp)
        movw %bx,-4(%ebp)
        movw %di,varSL.5
.L2:
        leal -16(%ebp),%esp
        popl %ebx
        popl %esi
        popl %edi
```

%0～%3に対応する，コンパイラが自動的に割り当てたレジスタ上の値を各変数にストアしている☒

リスト 7　インライン・アセンブラでの MMX，SSE/SSE2命令の使用例

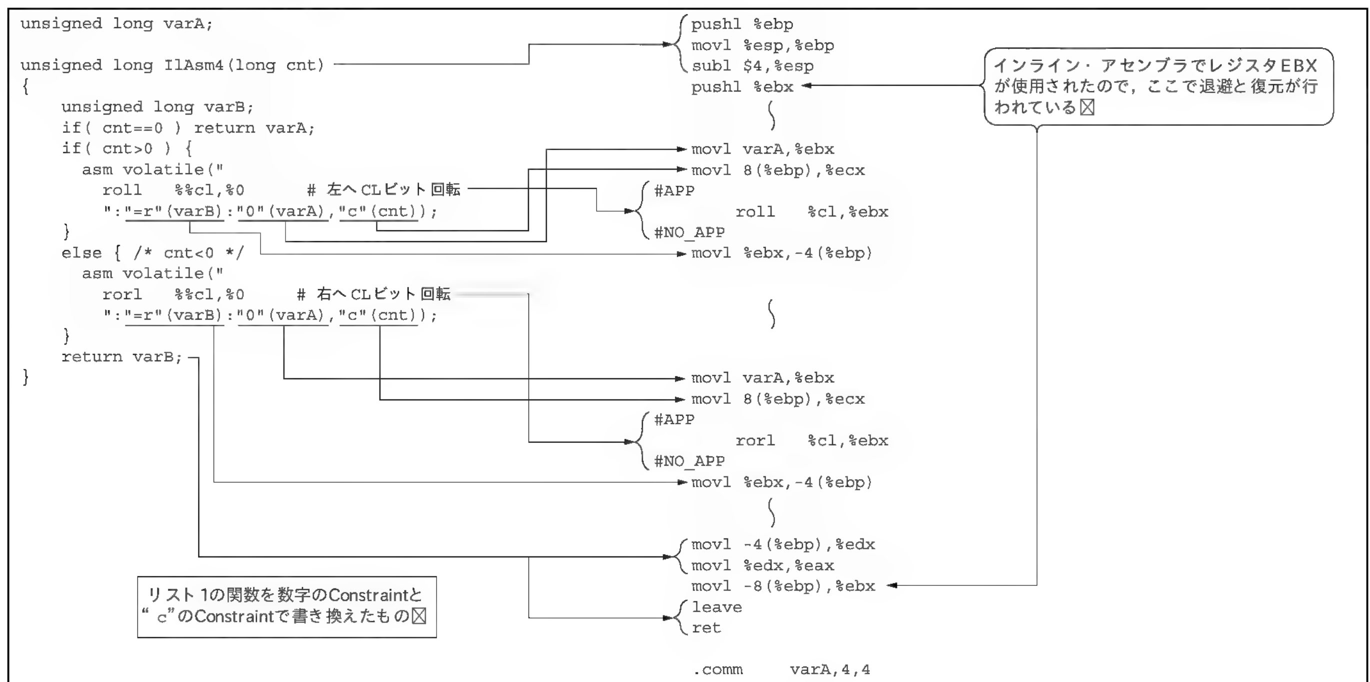


```
unsigned short *w4pakAdd(
  unsigned short *uspA,unsigned short *uspB
){
    static unsigned short w4pak_x[4];
    asm volatile("
      movq    (%1),%%mm0
      paddw   (%2),%%mm0
      movq    %%mm0,%0
      emms
      " : "=m"(w4pak_x)
        : "r"(uspA),"r"(uspB)
        : "%st","%st(1)","%st(2)","%st(3)",
          "%st(4)","%st(5)","%st(6)","%st(7)");
    return w4pak_x;
}

float *f4pakAdd(
  float *fpA,float *fpB
){
    static float f4pak_x[4];
    asm volatile("
      movups    (%1),%%xmm0
      movups    (%2),%%xmm1
      addps     %%xmm1,%%xmm0
      movups    %%xmm0,%0
      " : "=m"(f4pak_x)
        : "r"(fpA),"r"(fpB)
        : "%xmm0","%xmm1");
    return f4pak_x;
}

double *d2pakAdd(
  double *dpA,double *dpB
){
    static double d2pak_x[2];
    asm volatile("
      movupd    (%1),%%xmm0
      movupd    (%2),%%xmm1
      addpd     %%xmm1,%%xmm0
      movupd    %%xmm0,%0
      " : "=m"(d2pak_x)
        : "r"(dpA),"r"(dpB)
        : "%xmm0","%xmm1");
    return d2pak_x;
}
```

リスト 8　数字の Constraint の使用例



ロード，そして実行結果のレジスタからメモリへのストアといった前後処理をアセンブラで記述する必要があります．

そこで，このような場合に使われるのが表1にある数字のConstraintです．数字のConstraintは，入出力オペランドで %0,%1,%2,…にすでに割り当てられたレジスタやメモリを示しています．つまり，数字のConstraintを使うことで一つのオペランドに入力と出力の両方の属性をもたせることができるわけです．

これにより，めんどうな前後処理はコンパイラが自動的にコードを生成するため，目的のアセンブリ命令のみを記述することができるようになります．

リスト8はその使用例です．

● インライン・アセンブラにおけるFPU命令の使用

FPUの浮動小数点の演算命令は，スタック構造をしたレジスタ・スタックで演算を行います．スタックでの演算は，メモリとの値のやり取りや演算の実行で，スタック上の値とレジスタ番号の対応が動的に変わるため，値とレジスタの対応を把握するのがたいへんです．そのため，FPUの浮動小数点の演算命令を使用するには，つねに値とレジスタST(n)との対応を考えながらプログラミングする必要があります．

インライン・アセンブラでFPU命令の演算命令を使用する場合，このような理由から，どうしても変数の値をレジスタ・スタックにPUSHしたり，演算結果をレジスタ・スタックから変数にストアする前後処理が必要になります．

このようなスタックでの演算では，さきほど述べた数字のConstraintの機能だけでは，この前後処理を省くことはできません．

そこで使われるのが，レジスタ・スタックを表すConstraintのfとtとu，そしてModifier Charactersの&です．これらを使用することで，FPUの浮動小数点の演算命令でも，目的の演算命令を，前後処理を記述することなしに使用することができます．

Constraintのfは，コンパイラが決めたレジスタ・スタックの任意のレジスタST(n)を表します．そして，Constraintのtが ST(0)を表し，uが ST(1)を表しています．

Modifier Charactersの&は，入力オペランドとして使用されますが，CPUの動作により内容が変更されることを示します．

その使用例としてリスト9に，FPUのFSINCOS命令とF2XM1命令，FYL2XP1命令のインライン・アセンブラでの記述例を示します．FSINCOS命令は，一つの命令で正弦(sin)と余弦(cos)を同時に求めることができる便利な命令です．そしてF2XM1命令は 2^x-1 を，FYL2XP1命令は $y \times \log_2(x+1)$ を求める超越関数命令で，誤差の少ない双曲線関数，逆双曲線関数を求めるときに使用します．

リスト9(a)は，Constraintのmを使用した一般的なインライン・アセンブラの記述です．そのため，事前にレジスタ・スタックへの変数値をPUSHし，演算結果をレジスタ・スタックから変数へストアするといった前後処理の記述が必要です．

これをConstraintのtとu，そしてModifier Charactersの&で書き換えたのがリスト9(b)です．このリストを見るとわかるように，Constraintのtとu，そしてModifier Charactersの&を使用することで，前後処理のコードはコンパイラが自動

リスト 9　FPUの浮動小数点命令の使用例

```
/* 戻り値 = sin(ang), *pbCos = cos(ang) */
double Fsincos_a(double ang,double *pdCos)
{
    double vSin,vCos;
    asm volatile("
      fldl     %2
      fsincos
      fstpl    %0
      fstpl    %1
      ":"=m"(vCos),"=m"(vSin) :"m"(ang)
       :"%st","%st(1)");
    if( pdCos ) *pdCos = vCos;
    return vSin;
}

/* 戻り値 = pow(2,x)-1 */
double F2xm1_a(double x)
{
    double z;
    asm volatile("
      fldl   %1
      f2xm1
      fstpl  %0
      ":"=m"(z) :"m"(x)
       :"%st");
    return z;
}

/* 戻り値 = y * log2(x+1) */
double Fyl2xp1_a(double y,double x)
{
    double z;
    asm volatile("
       fldl    %2
       fldl    %1
       fyl2xp1
       fstp    %0
      ":"=m"(z) :"m"(x),"m"(y)
       :"st","st(1)");
    return z;
}
```

（a）Constraintの“m”を使った記述

```
/* 戻り値 = sin(ang), *pbCos = cos(ang) */
double Fsincos_b(double ang,double *pdCos)
{
    double vSin,vCos;
    asm volatile("fsincos
      ":"=&t"(vCos),"=u"(vSin):"0"(ang));
    if( pdCos ) *pdCos = vCos;
    return vSin;
}

/* 戻り値 = pow(2,x)-1 */
double F2xm1_b(double x)
{
    double z;
    asm volatile("f2xm1
      ":"=&t"(z):"0"(x));
    return z;
}

/* 戻り値 = y * log2(x+1) */
double Fyl2xp1_b(double y,double x)
{
    double z;
    asm volatile("fyl2xp1
      ":"=&t"(z):"0"(x),"u"(y):"%st(1)");
    return z;
}
```

Constraintの"t"，"u"でFPUのレジスタST(0)，ST(1)を直接指定することで，ST(0)，ST(1)に対する変数とのロード/ストアが自動的に行われるようになる

このasm文では，入出力パラメータでST(0)，ST(1)の使用を指定している．しかし，fyl2xp1命令を実行すると結果はST(0)のみとなりST(1)は使用されない．
このような命令では命令実行後使用されないレジスタは変更されたレジスタと見なし「変更レジスタのフィールド」に指定する

（b）Constraint“t”，“u”と Modifier Characters“&”を使った記述

的に生成するため，プログラマは目的のFPU命令のみ記述することが可能となります．

＊　　　　＊

86系32ビットCPU，そしてWindowsのアセンブラMASM，Linuxのアセンブラgasについて解説してきた「開発技術者のためのアセンブラ入門」も，今回をもって最終回となりました．

長かったこの連載にお付き合いいただいた読者の皆さま，ありがとうございました．

おおぬき・ひろゆき　大貫ソフトウェア設計事務所

フリーソフトウェア徹底活用講座

第19回 GCC2.95から追加変更のあったオプションの補足と検証(その7)

岸 哲夫

今回も引き続きGCC2.95から追加変更のあったオプションの補足と検証を行います．重要な「最適化オプション」について扱います．(筆者)

● -floop-optimize

ループの最適化を実行します．ループ内部の定数式を移動させたり，出口テスト条件を単純化したり，演算子の強さの低減を行い，ループの展開を行います．

オプションの指定の方法は次のとおりです．

● 指定あり

```
gcc test222.c -S  -O3  -floop-optimize
```

● 指定なし

```
gcc test222.c -S  -O3  -fno-loop-optimize
```

ソースと生成されたコードをリスト1～リスト3に示します．

最適化の結果，最初のループは削除されず，空のループになり，2番目のループは削除されています．サイズも小さくなり，処理速度も速くなっています．

● -fmerge-all-constants，-fmerge-constants

同一の定数および同一の変数を合併することを試みます．二つのオプションの意味はほぼ同一です．-fmerge-all-constantsの場合，一定の初期化された配列や，浮動小数点型の変数に対しても試みます．

オプションの指定の方法は次のとおりです．

リスト1　-floop-optimizeオプションを使う例(test222.c)

```
//ループ内最適化の例
#include      <stdio.h>
int main()
{
    int  xx;
    int  ix;
    int  xx1  =    100;
    int  xx2  =    10;
    xx        =    xx1  * xx2;
    for(ix=0;ix<10;ix++)
    {
         xx   =    100;
    }
    xx        =    0;
    for(ix=0;ix<4;ix++)
    {
         xx1  =    200;
         xx   =    xx1  +xx;
    }
    return    xx2  *    20;
}
```

リスト2　-floop-optimizeオプションを付けて生成されたアセンブラ・ソース(test222a.s)

```
    .file     "test222.c"
    .text
    .p2align 2,,3
.globl main
    .type     main,     @function
main:
    pushl     %ebp
    movl      %esp,     %ebp
    subl      $8,       %esp
    andl      $-16,     %esp
    movl      $9,       %eax
    .p2align 2,,3
.L6:
    decl      %eax
    jns       .L6
    movl      $200,     %eax
    leave
    ret
    .size     main, .-main
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト3　-fno-loop-optimizeオプションを付けて生成されたアセンブラ・ソース(test222.s)

```
    .file     "test222.c"
    .text
    .p2align 2,,3
.globl main
    .type     main,     @function
main:
    pushl     %ebp
    movl      %esp,     %ebp
    subl      $8,       %esp
    andl      $-16,     %esp
    xorl      %eax,     %eax
    .p2align 2,,3
.L6:
    incl      %eax
    cmpl      $9,       %eax
    jle  .L6
    xorl      %eax,     %eax
    .p2align 2,,3
.L11:
    incl      eax
    cmpl      $3,       %eax
    jle  .L11
    movl      $200,     %eax
    leave
    ret
    .size     main,     .-main
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト 4　-fmerge-all-constants オプションを使う例(test223.c)

```
//fmerge-constantsの例
#include <math.h>
const float f_d1 = 4.672299837f;
const float f_d2 = 4.672299837f;

float test(float f1)
{
    return f1 * f_d1 / f_d2;
}
int main()
{
    float    xx;
    xx   =   test(0.25);
    return 0;
}
```

● -fmerge-all-constants 指定あり

gcc test223.c -o test223 -O2 -fmerge-all-constants -fomit-frame-pointer

● -fmerge-all-constants 指定なし

gcc test223.c -o test223 -O2 -fmerge-constants -fomit-frame-pointer

ソースと生成されたコードを，リスト 4〜リスト 6に示します．

なお，オブジェクト・ダンプは，objdump -Dで取得しています．

リスト 5　-fmerge-all-constants オプションを付けて生成されたオブジェクト・ダンプ・リスト(test223.txt)

```
test223:     ファイル形式 elf32-i386

セクション .interp の逆アセンブル:

08048114 <.interp>:
 8048114:    2f                    das
 8048115:    6c                    insb   (%dx),%es:(%edi)
         ----------省略-------------
 8048124:    2e 32 00              xor    %cs:(%eax),%al
セクション .note.ABI-tag の逆アセンブル:

08048128 <.note.ABI-tag>:
 8048128:    04 00                 add    $0x0,%al
         ----------省略-------------
 8048145:    00 00                 add    %al,(%eax)
    ...
セクション .hash の逆アセンブル:

08048148 <.hash>:
 8048148:    03 00                 add    (%eax),%eax
         ----------省略-------------
 804816d:    00 00                 add    %al,(%eax)
    ...
セクション .dynsym の逆アセンブル:

08048170 <.dynsym>:
    ...
 8048180:    3d 00 00 00 00        cmp    $0x0,%eax
         ----------省略-------------
 80481bd:    00 00                 add    %al,(%eax)
    ...
セクション .dynstr の逆アセンブル:

080481c0 <.dynstr>:
 80481c0:    00 5f 4a              add    %bl,0x4a(%edi)
         ----------省略-------------
 8048217:    30 00                 xor    %al,(%eax)
セクション .gnu.version の逆アセンブル:

0804821a <.gnu.version>:
 804821a:    00 00                 add    %al,(%eax)
 804821c:    02 00                 add    (%eax),%al
 804821e:    01 00                 add    %eax,(%eax)
 8048220:    00 00                 add    %al,(%eax)
    ...
セクション .gnu.version_r の逆アセンブル:

08048224 <.gnu.version_r>:
 8048224:    01 00                 add    %eax,(%eax)
         ----------省略-------------
 8048241:    00 00                 add    %al,(%eax)
    ...
セクション .rel.dyn の逆アセンブル:

08048244 <.rel.dyn>:
 8048244:    14 95                 adc    $0x95,%al
 8048246:    04 08                 add    $0x8,%al
 8048248:    06                    push   %es
 8048249:    04 00                 add    $0x0,%al
    ...
セクション .rel.plt の逆アセンブル:

0804824c <.rel.plt>:
 804824c:    24 95                 and    $0x95,%al
 804824e:    04 08                 add    $0x8,%al
 8048250:    07                    pop    %es
 8048251:    01 00                 add    %eax,(%eax)
    ...
セクション .init の逆アセンブル:

08048254 <_init>:
 8048254:    55                    push   %ebp
 8048255:    89 e5                 mov    %esp,%ebp
 8048257:    83 ec 08              sub    $0x8,%esp
 804825a:    e8 51 00 00 00        call   80482b0
                                             <call_gmon_start>
 804825f:    e8 ac 00 00 00        call   8048310
                                               <frame_dummy>
 8048264:    e8 7f 01 00 00        call   80483e8
                                        <__do_global_ctors_aux>
 8048269:    c9                    leave
 804826a:    c3                    ret
セクション .plt の逆アセンブル:

0804826c <.plt>:
 804826c:    ff 35 1c 95 04 08     pushl  0x804951c
         ----------省略-------------
 8048287:    e9 e0 ff ff ff        jmp    804826c <_init+0x18>
セクション .text の逆アセンブル:

0804828c <_start>:
 804828c:    31 ed                 xor    %ebp,%ebp
 804828e:    5e                    pop    %esi
 804828f:    89 e1                 mov    %esp,%ecx
 8048291:    83 e4 f0              and    $0xfffffff0,%esp
 8048294:    50                    push   %eax
 8048295:    54                    push   %esp
 8048296:    52                    push   %edx
 8048297:    68 a4 83 04 08        push   $0x80483a4
 804829c:    68 5c 83 04 08        push   $0x804835c
 80482a1:    51                    push   %ecx
 80482a2:    56                    push   %esi
 80482a3:    68 4c 83 04 08        push   $0x804834c
 80482a8:    e8 cf ff ff ff        call   804827c <_init+0x28>
 80482ad:    f4                    hlt
 80482ae:    90                    nop
 80482af:    90                    nop

080482b0 <call_gmon_start>:
 80482b0:    55                    push   %ebp
 80482b1:    89 e5                 mov    %esp,%ebp
 80482b3:    53                    push   %ebx
 80482b4:    e8 00 00 00 00        call   80482b9
                                        <call_gmon_start+0x9>
 80482b9:    5b                    pop    %ebx
 80482ba:    81 c3 5f 12 00 00     add    $0x125f,%ebx
 80482c0:    50                    push   %eax
 80482c1:    8b 83 fc ff ff ff     mov
                                       0xfffffffc(%ebx),%eax
 80482c7:    85 c0                 test   %eax,%eax
 80482c9:    74 02                 je     80482cd
```

リスト 5 -fmerge-all-constants オプションを付けて生成されたオブジェクト・ダンプ・リスト(test223.txt)(つづき)

```
                                               <call_gmon_start+0x1d>
 80482cb:     ff d0                      call   *%eax
 80482cd:     8b 5d fc                   mov
                                                 0xfffffffc(%ebp),%ebx
 80482d0:     c9                         leave
 80482d1:     c3                         ret
 80482d2:     90                         nop
 80482d3:     90                         nop

080482d4 <__do_global_dtors_aux>:
 80482d4:     55                         push   %ebp
 80482d5:     89 e5                      mov    %esp,%ebp
 80482d7:     83 ec 08                   sub    $0x8,%esp
 80482da:     80 3d 34 95 04 08 00       cmpb   $0x0,0x8049534
 80482e1:     75 29                      jne    804830c
                                        <__do_global_dtors_aux+0x38>
 80482e3:     a1 30 95 04 08             mov    0x8049530,%eax
 80482e8:     8b 10                      mov    (%eax),%edx
 80482ea:     85 d2                      test   %edx,%edx
 80482ec:     74 17                      je     8048305
                                        <__do_global_dtors_aux+0x31>
 80482ee:     89 f6                      mov    %esi,%esi
 80482f0:     83 c0 04                   add    $0x4,%eax
 80482f3:     a3 30 95 04 08             mov    %eax,0x8049530
 80482f8:     ff d2                      call   *%edx
 80482fa:     a1 30 95 04 08             mov    0x8049530,%eax
 80482ff:     8b 10                      mov    (%eax),%edx
 8048301:     85 d2                      test   %edx,%edx
 8048303:     75 eb                      jne    80482f0
                                        <__do_global_dtors_aux+0x1c>
 8048305:     c6 05 34 95 04 08 01       movb   $0x1,0x8049534
 804830c:     c9                         leave
 804830d:     c3                         ret
 804830e:     89 f6                      mov    %esi,%esi

08048310 <frame_dummy>:
 8048310:     55                         push   %ebp
              ----------省略-------------
 804833b:     90                         nop

0804833c <test>:
 804833c:     d9 05 30 84 04 08          flds   0x8048430
 8048342:     d9 44 24 04                flds   0x4(%esp)
 8048346:     d8 c9                      fmul   %st(1),%st
 8048348:     de f1                      fdivp  %st,%st(1)
 804834a:     c3                         ret
 804834b:     90                         nop

0804834c <main>:
 804834c:     55                         push   %ebp
 804834d:     89 e5                      mov    %esp,%ebp
 804834f:     83 ec 08                   sub    $0x8,%esp
 8048352:     83 e4 f0                   and    $0xfffffff0,%esp
 8048355:     31 c0                      xor    %eax,%eax
 8048357:     c9                         leave
 8048358:     c3                         ret
 8048359:     90                         nop
 804835a:     90                         nop
 804835b:     90                         nop

0804835c <__libc_csu_init>:
 804835c:     55                         push   %ebp
              ----------省略-------------
 80483a3:     c3                         ret

080483a4 <__libc_csu_fini>:
 80483a4:     55                         push   %ebp
              ----------省略-------------
 80483e6:     eb e5                      jmp    80483cd
                                               <__libc_csu_fini+0x29>

080483e8 <__do_global_ctors_aux>:
 80483e8:     55                         push   %ebp
 80483e9:     89 e5                      mov    %esp,%ebp
 80483eb:     53                         push   %ebx
 80483ec:     52                         push   %edx
 80483ed:     a1 38 94 04 08             mov    0x8049438,%eax
 80483f2:     83 f8 ff                   cmp    $0xffffffff,%eax
 80483f5:     bb 38 94 04 08             mov    $0x8049438,%ebx
 80483fa:     74 0c                      je     8048408
                                        <__do_global_ctors_aux+0x20>
 80483fc:     83 eb 04                   sub    $0x4,%ebx
 80483ff:     ff d0                      call   *%eax
 8048401:     8b 03                      mov    (%ebx),%eax
 8048403:     83 f8 ff                   cmp    $0xffffffff,%eax
 8048406:     75 f4                      jne    80483fc
                                        <__do_global_ctors_aux+0x14>
 8048408:     58                         pop    %eax
 8048409:     5b                         pop    %ebx
 804840a:     c9                         leave
 804840b:     c3                         ret
セクション .fini の逆アセンブル:

0804840c <_fini>:
 804840c:     55                         push   %ebp
              ----------省略-------------
 8048426:     c3                         ret
セクション .rodata の逆アセンブル:

08048428 <_fp_hw>:
 8048428:     03 00                      add    (%eax),%eax
  ...

0804842c <_IO_stdin_used>:
 804842c:     01 00                      add    %eax,(%eax)
 804842e:     02 00                      add    (%eax),%al

08048430 <f_d1>:
 8048430:     7b 83                      jnp    80483b5
                                               <__libc_csu_fini+0x11>
 8048432:     95                         xchg   %eax,%ebp
 8048433:     40                         inc    %eax
セクション .eh_frame の逆アセンブル:

08048434 <__FRAME_END__>:
 8048434:     00 00                      add    %al,(%eax)
  ...
セクション .ctors の逆アセンブル:

08049438 <__CTOR_LIST__>:
 8049438:     ff                         (bad)
 8049439:     ff                         (bad)
 804943a:     ff                         (bad)
 804943b:     ff 00                      incl   (%eax)

0804943c <__CTOR_END__>:
 804943c:     00 00                      add    %al,(%eax)
  ...
セクション .dtors の逆アセンブル:

08049440 <__DTOR_LIST__>:
 8049440:     ff                         (bad)
 8049441:     ff                         (bad)
 8049442:     ff                         (bad)
 8049443:     ff 00                      incl   (%eax)

08049444 <__DTOR_END__>:
 8049444:     00 00                      add    %al,(%eax)
  ...
セクション .jcr の逆アセンブル:

08049448 <__JCR_END__>:
 8049448:     00 00                      add    %al,(%eax)
  ...
セクション .dynamic の逆アセンブル:

0804944c <_DYNAMIC>:
 804944c:     01 00                      add    %eax,(%eax)
              ----------省略-------------
 80494e2:     04 08                      add    $0x8,%al
  ...
セクション .got の逆アセンブル:

08049514 <.got>:
 8049514:     00 00                      add    %al,(%eax)
  ...
セクション .got.plt の逆アセンブル:

08049518 <_GLOBAL_OFFSET_TABLE_>:
 8049518:     4c                         dec    %esp
 8049519:     94                         xchg   %eax,%esp
 804951a:     04 08                      add    $0x8,%al
  ...
 8049524:     82                         (bad)
```

リスト5　-fmerge-all-constants オプションを付けて生成されたオブジェクト・ダンプ・リスト(test223.txt)(つづき)

```
 8049525:     82                          (bad)
 8049526:     04 08                       add     $0x8,%al
セクション .data の逆アセンブル:

08049528 <__data_start>:
 8049528:     00 00                       add     %al,(%eax)
    ...

0804952c <__dso_handle>:
 804952c:     00 00                       add     %al,(%eax)
    ...

08049530 <p.0>:
 8049530:     44                          inc     %esp
 8049531:     94                          xchg    %eax,%esp
 8049532:     04 08                       add     $0x8,%al
セクション .bss の逆アセンブル:

08049534 <completed.1>:
 8049534:     00 00                       add     %al,(%eax)
    ...
セクション .comment の逆アセンブル:

00000000 <.comment>:
   0:    00 47 43                         add     %al,0x43(%edi)
         ----------省略-------------
 12f:    33 29                            xor     (%ecx),%ebp
    ...
```

リスト6　-fno-merge-all-constants オプションを付けて生成されたオブジェクト・ダンプ・リスト(test223a.txt)

```
test223:     ファイル形式 elf32-i386

セクション .interp の逆アセンブル:

08048114 <.interp>:
 8048114:     2f                          das
 8048115:     6c                          insb    (%dx),%es:(%edi)
         ----------省略-------------
 8048124:     2e 32 00                    xor     %cs:(%eax),%al
セクション .note.ABI-tag の逆アセンブル:

08048128 <.note.ABI-tag>:
 8048128:     04 00                       add     $0x0,%al
         ----------省略-------------
 8048145:     00 00                       add     %al,(%eax)
    ...
セクション .hash の逆アセンブル:

08048148 <.hash>:
 8048148:     03 00                       add     (%eax),%eax
         ----------省略-------------
 804816d:     00 00                       add     %al,(%eax)
    ...
セクション .dynsym の逆アセンブル:

08048170 <.dynsym>:
    ...
 8048180:     3d 00 00 00 00              cmp     $0x0,%eax
         ----------省略-------------
 80481bd:     00 00                       add     %al,(%eax)
    ...
セクション .dynstr の逆アセンブル:

080481c0 <.dynstr>:
 80481c0:     00 5f 4a                    add     %bl,0x4a(%edi)
         ----------省略-------------
 8048217:     30 00                       xor     %al,(%eax)
セクション .gnu.version の逆アセンブル:

0804821a <.gnu.version>:
 804821a:     00 00                       add     %al,(%eax)
 804821c:     02 00                       add     (%eax),%al
 804821e:     01 00                       add     %eax,(%eax)
 8048220:     00 00                       add     %al,(%eax)
    ...
セクション .gnu.version_r の逆アセンブル:

08048224 <.gnu.version_r>:
 8048224:     01 00                       add     %eax,(%eax)
         ----------省略-------------
 8048241:     00 00                       add     %al,(%eax)
    ...
セクション .rel.dyn の逆アセンブル:

08048244 <.rel.dyn>:
 8048244:     1c 95                       sbb     $0x95,%al
 8048246:     04 08                       add     $0x8,%al
 8048248:     06                          push    %es
 8048249:     04 00                       add     $0x0,%al
    ...
セクション .rel.plt の逆アセンブル:
0804824c <.rel.plt>:
 804824c:     2c 95                       sub     $0x95,%al
 804824e:     04 08                       add     $0x8,%al
 8048250:     07                          pop     %es
 8048251:     01 00                       add     %eax,(%eax)
    ...
セクション .init の逆アセンブル:

08048254 <_init>:
 8048254:     55                          push    %ebp
 8048255:     89 e5                       mov     %esp,%ebp
 8048257:     83 ec 08                    sub     $0x8,%esp
 804825a:     e8 51 00 00 00              call    80482b0
                                                   <call_gmon_start>
 804825f:     e8 ac 00 00 00              call    8048310
                                                       <frame_dummy>
 8048264:     e8 7f 01 00 00              call    80483e8
                                             <__do_global_ctors_aux>
 8048269:     c9                          leave
 804826a:     c3                          ret
セクション .plt の逆アセンブル:

0804826c <.plt>:
 804826c:     ff 35 24 95 04 08           pushl   0x8049524
         ----------省略-------------
 8048287:     e9 e0 ff ff ff              jmp     804826c
                                                        <_init+0x18>
セクション .text の逆アセンブル:

0804828c <_start>:
 804828c:     31 ed                       xor     %ebp,%ebp
 804828e:     5e                          pop     %esi
 804828f:     89 e1                       mov     %esp,%ecx
 8048291:     83 e4 f0                    and     $0xfffffff0,%esp
 8048294:     50                          push    %eax
 8048295:     54                          push    %esp
 8048296:     52                          push    %edx
 8048297:     68 a4 83 04 08              push    $0x80483a4
 804829c:     68 5c 83 04 08              push    $0x804835c
 80482a1:     51                          push    %ecx
 80482a2:     56                          push    %esi
 80482a3:     68 4c 83 04 08              push    $0x804834c
 80482a8:     e8 cf ff ff ff              call    804827c
                                                        <_init+0x28>
 80482ad:     f4                          hlt
 80482ae:     90                          nop
 80482af:     90                          nop

080482b0 <call_gmon_start>:
 80482b0:     55                          push    %ebp
 80482b1:     89 e5                       mov     %esp,%ebp
 80482b3:     53                          push    %ebx
 80482b4:     e8 00 00 00 00              call    80482b9
                                               <call_gmon_start+0x9>
 80482b9:     5b                          pop     %ebx
 80482ba:     81 c3 67 12 00 00           add     $0x1267,%ebx
 80482c0:     50                          push    %eax
 80482c1:     8b 83 fc ff ff ff           mov
                                              0xfffffffc(%ebx),%eax
 80482c7:     85 c0                       test    %eax,%eax
 80482c9:     74 02                       je      80482cd
                                              <call_gmon_start+0x1d>
 80482cb:     ff d0                       call    *%eax
```

リスト 6 -fno-merge-all-constants オプションを付けて生成されたオブジェクト・ダンプ・リスト(test223a.txt)(つづき)

```
 80482cd:	8b 5d fc             	mov
                                    0xfffffffc(%ebp),%ebx
 80482d0:	c9                   	leave
 80482d1:	c3                   	ret
 80482d2:	90                   	nop
 80482d3:	90                   	nop

080482d4 <__do_global_dtors_aux>:
 80482d4:	55                   	push   %ebp
 80482d5:	89 e5                	mov    %esp,%ebp
 80482d7:	83 ec 08             	sub    $0x8,%esp
 80482da:	80 3d 3c 95 04 08 00 	cmpb   $0x0,0x804953c
 80482e1:	75 29                	jne    804830c
                              <__do_global_dtors_aux+0x38>
 80482e3:	a1 38 95 04 08       	mov    0x8049538,%eax
 80482e8:	8b 10                	mov    (%eax),%edx
 80482ea:	85 d2                	test   %edx,%edx
 80482ec:	74 17                	je     8048305
                              <__do_global_dtors_aux+0x31>
 80482ee:	89 f6                	mov    %esi,%esi
 80482f0:	83 c0 04             	add    $0x4,%eax
 80482f3:	a3 38 95 04 08       	mov    %eax,0x8049538
 80482f8:	ff d2                	call   *%edx
 80482fa:	a1 38 95 04 08       	mov    0x8049538,%eax
 80482ff:	8b 10                	mov    (%eax),%edx
 8048301:	85 d2                	test   %edx,%edx
 8048303:	75 eb                	jne    80482f0
                              <__do_global_dtors_aux+0x1c>
 8048305:	c6 05 3c 95 04 08 01 	movb   $0x1,0x804953c
 804830c:	c9                   	leave
 804830d:	c3                   	ret
 804830e:	89 f6                	mov    %esi,%esi

08048310 <frame_dummy>:
 8048310:	55                   	push   %ebp
          ----------省略-------------
 804833b:	90                   	nop

0804833c <test>:
 804833c:	d9 05 38 84 04 08    	flds   0x8048438
 8048342:	d9 44 24 04          	flds   0x4(%esp)
 8048346:	d8 c9                	fmul   %st(1),%st
 8048348:	de f1                	fdivp  %st,%st(1)
 804834a:	c3                   	ret
 804834b:	90                   	nop

0804834c <main>:
 804834c:	55                   	push   %ebp
 804834d:	89 e5                	mov    %esp,%ebp
 804834f:	83 ec 08             	sub    $0x8,%esp
 8048352:	83 e4 f0             	and    $0xfffffff0,%esp
 8048355:	31 c0                	xor    %eax,%eax
 8048357:	c9                   	leave
 8048358:	c3                   	ret
 8048359:	90                   	nop
 804835a:	90                   	nop
 804835b:	90                   	nop

0804835c <__libc_csu_init>:
 804835c:	55                   	push   %ebp
          ----------省略-------------
 80483a3:	c3                   	ret

080483a4 <__libc_csu_fini>:
 80483a4:	55                   	push   %ebp
          ----------省略-------------
 80483e6:	eb e5                	jmp    80483cd
                                  <__libc_csu_fini+0x29>

080483e8 <__do_global_ctors_aux>:
 80483e8:	55                   	push   %ebp
 80483e9:	89 e5                	mov    %esp,%ebp
 80483eb:	53                   	push   %ebx
 80483ec:	52                   	push   %edx
 80483ed:	a1 40 94 04 08       	mov    0x8049440,%eax
 80483f2:	83 f8 ff             	cmp    $0xffffffff,%eax
 80483f5:	bb 40 94 04 08       	mov    $0x8049440,%ebx
 80483fa:	74 0c                	je     8048408
                              <__do_global_ctors_aux+0x20>
 80483fc:	83 eb 04             	sub    $0x4,%ebx
 80483ff:	ff d0                	call   *%eax
 8048401:	8b 03                	mov    (%ebx),%eax
 8048403:	83 f8 ff             	cmp    $0xffffffff,%eax
 8048406:	75 f4                	jne    80483fc
                              <__do_global_ctors_aux+0x14>
 8048408:	58                   	pop    %eax
 8048409:	5b                   	pop    %ebx
 804840a:	c9                   	leave
 804840b:	c3                   	ret
セクション .fini の逆アセンブル:

0804840c <_fini>:
 804840c:	55                   	push   %ebp
          ----------省略-------------
 8048426:	c3                   	ret
セクション .rodata の逆アセンブル:

08048428 <_fp_hw>:
 8048428:	03 00                	add    (%eax),%eax
	...

0804842c <_IO_stdin_used>:
 804842c:	01 00                	add    %eax,(%eax)
 804842e:	02 00                	add    (%eax),%al

08048430 <f_d1>:
 8048430:	7b 83                	jnp    80483b5
                                  <__libc_csu_fini+0x11>
 8048432:	95                   	xchg   %eax,%ebp
 8048433:	40                   	inc    %eax

08048434 <f_d2>:
 8048434:	7b 83                	jnp    80483b9
                                  <__libc_csu_fini+0x15>
 8048436:	95                   	xchg   %eax,%ebp
 8048437:	40                   	inc    %eax
 8048438:	7b 83                	jnp    80483bd
                                  <__libc_csu_fini+0x19>
 804843a:	95                   	xchg   %eax,%ebp
 804843b:	40                   	inc    %eax
セクション .eh_frame の逆アセンブル:

0804843c <__FRAME_END__>:
 804843c:	00 00                	add    %al,(%eax)
	...
セクション .ctors の逆アセンブル:

08049440 <__CTOR_LIST__>:
 8049440:	ff                   	(bad)
 8049441:	ff                   	(bad)
 8049442:	ff                   	(bad)
 8049443:	ff 00                	incl   (%eax)

08049444 <__CTOR_END__>:
 8049444:	00 00                	add    %al,(%eax)
	...
セクション .dtors の逆アセンブル:

08049448 <__DTOR_LIST__>:
 8049448:	ff                   	(bad)
 8049449:	ff                   	(bad)
 804944a:	ff                   	(bad)
 804944b:	ff 00                	incl   (%eax)

0804944c <__DTOR_END__>:
 804944c:	00 00                	add    %al,(%eax)
	...
セクション .jcr の逆アセンブル:

08049450 <__JCR_END__>:
 8049450:	00 00                	add    %al,(%eax)
	...
セクション .dynamic の逆アセンブル:

08049454 <_DYNAMIC>:
 8049454:	01 00                	add    %eax,(%eax)
          ----------省略-------------
 80494ea:	04 08                	add    $0x8,%al
	...
セクション .got の逆アセンブル:

0804951c <.got>:
 804951c:	00 00                	add    %al,(%eax)
	...
```

リスト 6　-fno-merge-all-constants オプションを付けて生成されたオブジェクト・ダンプ・リスト(test223a.txt)(つづき)

```
セクション .got.plt の逆アセンブル:

08049520 <_GLOBAL_OFFSET_TABLE_>:
 8049520:      54                          push   %esp
 8049521:      94                          xchg   %eax,%esp
 8049522:      04 08                       add    $0x8,%al
    ...
 804952c:      82                          (bad)
 804952d:      82                          (bad)
 804952e:      04 08                       add    $0x8,%al
セクション .data の逆アセンブル:

08049530 <__data_start>:
 8049530:      00 00                       add    %al,(%eax)
    ...

08049534 <__dso_handle>:
 8049534:      00 00                       add    %al,(%eax)
    ...

08049538 <p.0>:
 8049538:      4c                          dec    %esp
 8049539:      94                          xchg   %eax,%esp
 804953a:      04 08                       add    $0x8,%al
セクション .bss の逆アセンブル:

0804953c <completed.1>:
 804953c:      00 00                       add    %al,(%eax)
    ...
セクション .comment の逆アセンブル:

00000000 <.comment>:
   0:    00 47 43                          add    %al,0x43(%edi)
         ----------省略-------------
 12f:    33 29                             xor    (%ecx),%ebp
    ...
```

リスト 7　-fmove-all-movables オプションを使う例(test224.c)

```
//ループ中の不変式計算すべてをループ外に移動する例
#include      <stdio.h>
int main()
{
    int  xx;
    int  ix;
    int  xx1  =     100;
    int  xx2  =     10;
    int  xx3;
    for(ix=0;ix<10;ix++)
    {
        int x1;
        x1   =    20;
        xx   =    xx1  * xx2;
        printf("%d¥n",ix+xx);
    }
}
```

リスト 8　-fmove-all-movables オプションを付けて生成されたアセンブラ・ソース(test224a.s)

```
    .file     "test224.c"
    .section  .rodata.str1.1,"aMS",@progbits,1
.LC0:
    .string   "%d¥n"
    .text
    .p2align 2,,3
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp,     %ebp
    pushl     %esi
    pushl     %ebx
    andl      $-16,     %esp
    movl      $1000,    %esi
    movl      $9,       %ebx
    .p2align 2,,3
.L6:
    subl      $8,       %esp
    pushl     %esi
    pushl     $.LC0
    call      printf
    incl      %esi
    addl      $16,      %esp
    decl      %ebx
    jns       .L6
    leal      -8(%ebp), %esp
    popl      %ebx
    popl      %esi
    leave
    ret
    .size     main,     .-main
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

const で指定された定数である f_d1，f_d2 は値が同一のため，オプションとして -fmerge-all-constants を付加すると f_d1 だけの領域しか確保されません．同一の値の変数でも別の名前で定義することは，プログラムの可読性を高めるためには有用だと思います．メモリ領域を気にしないのならば，このオプションを使う必要はありません．

● -fmove-all-movables

ループ中の不変式計算すべてをループ外に移動します．

● -freduce-all-givs

ループ中の一般誘導変数を削減します．

なお，-fmove-all-movables，および -freduce-all-givs のオプションについては，将来なくなることが決まっています．別のオプションが追加されるはずです．

オプションの指定の方法は次のとおりです．

●指定あり

```
gcc test224.c -S   -O3 -fmove-all-movables
                          -freduce-all-givs
```

●指定なし

```
gcc test224.c -S   -O3 -fno-move-all-movables
                          -fno-reduce-all-givs
```

ソースと生成されたコードをリスト 7～リスト 9に示します．

最適化の結果，コードの行数は増加しましたが，

```
xx = xx1 * xx2;
```

の式をループの外に移動し，xx1 が 100，xx2 が 10 なので，

```
xx = 1000
```

に置き換えられています．

● -fno-branch-count-reg

カウント・レジスタを使ってブランチ命令や減算命令を行いません．しかし，その代わりにレジスタ減算命令のシーケンスを作成します．このオプションは，それが可能な CPU 環境でサポートされます．

GCC 内部でアセンブラ・ソースを生成する時点で，CX レジスタ(カウント・レジスタ)をブランチ命令や，減算命令に割り当てないようなので，プログラム例は省略します．

● -fno-cprop-registers

レジスタ割り付けの前後の命令を分割します．その後，「コピーの伝搬」を行って命令文を減らします．

● -fno-default-inline

C++ のメンバ関数に関する最適化オプションです．ここでは触れません．

● -fno-defer-pop

関数呼び出しのたび，つねにその関数から戻るとすぐ引き数を POP します．関数呼び出しの後に引き数を POP しなければならない機種では，GCC はたいてい複数の関数呼び出しについてスタックに引き数を蓄積し，それらをすべて一度に POP します．

これはハードウェアのアドレスを共有して実行するような場合や，そのオブジェクトの永続性が短い関数から戻った場合などに指定するべきオプションです．

最適化レベル -O，-O2，-O3，-Os を指定している場合は，つねにこのオプションはオフになります．

● -fno-function-cse

関数のアドレスをレジスタに置かないようにします．ある関数を呼び出す命令は，それぞれ関数のアドレスを明示的に保持するようにします．

このオプションは効率の低いコードを生成しますが，実行中にほかのアセンブラ出力ソースを書き換えるようなことを行う場合には，このオプションを使用しないと混乱してしまいます．

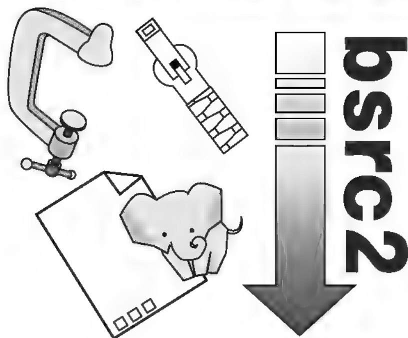
＊　　　＊

次回も続けて「最適化オプション」の続きを説明します．

きし・てつお

リスト 9　-fmove-all-movables オプションを付けて生成されたアセンブラ・ソース（test224.s）

```
    .file    "test224.c"
    .section .rodata.str1.1,"aMS",@progbits,1
.LC0:
    .string  "%d¥n"
    .text
    .p2align 2,,3
.globl main
    .type    main, @function
main:
    pushl    %ebp
    movl     %esp,    %ebp
    pushl    %ebx
    pushl    %eax
    andl     $-16,    %esp
    xorl     %ebx,    %ebx
    .p2align 2,,3
.L6:
    subl     $8,  %esp
    leal     1000(%ebx),   %edx
    pushl    %edx
    pushl    $.LC0
    incl     %ebx
    call     printf
    addl     $16, %esp
    cmpl     $9, %ebx
    jle      .L6
    movl     -4(%ebp),     %ebx
    leave
    ret
    .size    main,    .-main
    .section .note.GNU-stack,"",@progbits
    .ident   "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```



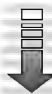
マルチキー・クイック・ソートと0-1-2 codingにより高速化と高圧縮率を実現した

高性能圧縮ツールbsrcの改良bsrc2（前編）

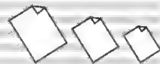
高速なソート・アルゴリズム，マルチキー・クイック・ソートについて

広井 誠

ファイルを圧縮する場合，LHA，zip，gzipなどが一般に使用されている．圧縮ツールで用いられているデータ圧縮アルゴリズムであるブロック・ソート(BlockSorting)は圧縮性能が優れている方法で，これを用いた圧縮ツールbzip2[10]はLHA，zip，gzipよりも高い圧縮率になる．
筆者は本誌2003年12月号と2004年1月号で，ブロック・ソートとレンジ・コーダ(RangeCoder)を用いた圧縮ツールbsrcを作成した．bsrcは，bzip2と同程度の圧縮率を達成したが，処理時間と圧縮率には改良の余地があった．そこで，これらの改良を行ったプログラムbsrc2を作ったところ，処理時間はbsrcよりも大幅に短縮し，圧縮率はbzip2を上回り，zzip[11]に匹敵する性能を実現できた．本稿では，このbsrc2について解説を行う．
（筆者）



bsrcからの改良点



●処理時間の改善——三分割法とマルチキー・クイック・ソート

bsrcは高い圧縮率を実現することができましたが，いくつかの課題が残されていました．その一つが処理時間です．ブロック・ソートはソート処理が必要になりますが，一般にソートはとても時間がかかる処理です．処理時間を短縮するため，先頭2記号で分布数えソートを行い，そのあとでマージ・ソートやクイック・ソートでしあげるなど，いろいろなくふうが考えられますが，もっと高速にソートできる方法があります．

巨大なテキスト・データを高速に検索するためのデータ構造には，suffix arrayがあります（後述）．suffix arrayの構築にはブロック・ソートと同様にデータのソートが必要になるため，高速なアルゴリズムが研究・開発されています．これらのアルゴリズムをブロック・ソートに適用することで，巨大なデータでも比較的短時間でソートすることができます．

Suffix arrayの構築法として有名な方法は，Larrson，Sadakane法，二段階ソート法（Two-Stage Sort），Copy Methodなどがあります．この中でCopy Methodはbzip2のブロック・ソートに使われています．bsrcは二段階ソート法とマージ・ソートを使っていますが，圧縮処理はbzip2よりも時間がかかります．bsrcが遅い理由として，文字列のソートにマージ・ソートを使っていることが挙げられ，これを改良することで，さらなる高速化が可能になります．

文字列のソートには「マルチキー・クイック・ソート（Multikey Quicksort）」というアルゴリズムがあり，クイック・ソートやマージ・ソートよりも高速にソートすることができます．そして，bsrcで用いた二段階ソートを改良した「三分割法」とマルチキー・クイック・ソートを用いることで，ブロック・ソートの処理時間を大幅に短縮することができます．

●圧縮率の改善——0-1-2 coding

もう一つの改良点が圧縮率です．bsrcではランレングス，MTF法，情報源モデルを改良して圧縮率を向上させていました．情報源モデルの改良では，記号をfirst codeとsecond codeの二つに分け，first codeの値によりsecond codeの出現頻度表を切り替えています．このようなモデルをstructured modelと呼びます．

このほかに，ブロック・ソートに適した情報源モデルに0-1-2 codingがあります．0-1-2 codingはstructured modelと組み合わることで，とても高い効果を発揮します．さらに，「混合法」という方法を適用すると，圧縮率を大幅に向上させることができます．

そこで以下に，bsrc2の改良点である，マルチキー・クイック・ソートと二段階ソート法，0-1-2 codingと混合法について解説します．



suffix arrayとブロック・ソート



まず最初に，マルチキー・クイック・ソートと三分割法を使ううえで前提となるsuffix arrayについて簡単に説明します．suffix arrayは1993年にManberとMyersにより提案されたデータ構造で，大規模なテキスト・データを高速に検索するために用いられます．

「サフィックス（suffix：接尾辞）」とは，記号列のある位置から末尾までの記号列のことです．たとえば，記号列"abcd"のサフィックスはabcd，bcd，cd，dの四つになります．suffix arrayはサフィックスを辞書順に並べた配列のことです．

簡単な例を示します．記号列"aeadacab"のsuffix arrayを作成します．図1に示すように，サフィックスは記号列のある位置から末尾までの記号列なので，サフィックスの開始位置（インデックス）で表すことができます．そして，インデックスを配列indexに格納し，図1のようにindexをサフィックスでソートします．その結果がsuffix arrayとなります．

ここで，suffix arrayはデータの終端を考慮してソートすることに注意してください．一般に，suffix arrayは記号列の終わりに終端を表す特別な記号を付加してソートします．これに対し，ブロック・ソートはデータの変換が目的なので，終端記号を付け加える必要はありません．

このように，suffix arrayはサフィックスをソートした配列なので，記号列の検索は二分探索を使って高速に行うことがで

index	suffix
0	aeadacab☒
1	eadacab☒
2	adacab☒
3	dacab☒
4	acab☒
5	cab☒
6	ab☒
7	b

ソートする☒ →

index	suffix
6	ab☒
4	acab☒
2	adacab☒
0	aeadacab☒
7	b☒
5	cab☒
3	dacab☒
1	eadacab

suffix arrayは[6, 4, 2, 0, 7, 5, 3, 1]になる☒

図1　suffix array

リスト 1　マルチキー・クイック・ソート(疑似コード)

```
multikey-quicksort( low, high, depth )
{
  if( 区間のデータ数が NUM 以下 ){
    単純なソートアルゴリズムに切り替えてソート
  } else {
    depth 番目の文字で枢軸を選択して low - high を 3 分割する
    /* < : low - m1-1, = : m1 - m2-1, > : m2 - high */
    multikey-quicksort( low, m1 - 1, depth );
    if( depth 番目の文字 != '¥0' ){
      multikey-quicksort( m1, m2 - 1, depth + 1 );
    }
    multikey-quicksort( m2, high, depth );
  }
}
```

きます．欠点はsuffix arrayの作成に時間がかかることです．このため，高速なアルゴリズムが研究・開発されています．記号列を高速にソートできる方法としては，「マルチキー・クイック・ソート」とサフィックス間の関係を利用して，高速にsuffix arrayを構築する「二段階ソート法」があります．

マルチキー・クイック・ソートとは

クイック・ソートは汎用的なソート・アルゴリズムですが，1997年にJon Bentley氏とRobert Sedgewick氏が発表したマルチキー・クイック・ソート(Multikey Quicksort)(4)は文字列のソートに適した高速なアルゴリズムです．ここではC言語で取り扱う文字列を例にして，マルチキー・クイック・ソートについて説明します．

マルチキー・クイック・ソートで文字列をソートする場合，普通のクイック・ソートと大きく異なる点が二つあります．

一つは文字単位で比較を行うことです．文字列をソートする場合，一般的なソートは文字列単位で比較を行います．ところが，マルチキー・クイック・ソートは最初に1文字目を比較してソートを行い，ソートが完了しない場合(同じ値が複数ある場合)は，さらに2文字目を比較してソートを行う，というように先頭から順番に文字を比較してソートを行います．

もう一つは区間の分け方です．普通のクイック・ソートは枢軸を基準にして，小さいデータと大きいデータの二つの区間に分割します．これに対し，マルチキー・クイック・ソートは枢軸を基準にするところまではクイック・ソートと同じですが，区間を2分割するのではなく，小さいデータ，等しいデータ，大きいデータの3分割にするところが特徴です．

たとえば，n番目の文字を比較して区間を3分割したとしましょう．小さいデータと大きいデータの区間はn番目の文字でソートを続けます．これは普通のクイック・ソートと同じです．枢軸と等しいデータが複数ある場合，ソートはまだ完了していません．この区間の文字列は，$n+1$番目の文字を比較してソートを続けます．このように，マルチキー・クイック・ソートは枢軸と等しいデータを集め，その区間は次の文字を比較することで文字列をソートします．

このアルゴリズムを疑似コードでプログラムすると，リスト1のようになります．

マルチキー・クイック・ソートは再帰呼び出しを使うと簡単にプログラムできます．区間を`low`と`high`で表し，比較する文字の位置を`depth`で表します．区間のデータ数が一定の個数(`NUM`)以下になったら，単純なソート・アルゴリズムに切り替えます．このほうが少しだけ速くなります．

`depth`番目の文字で区間を3分割したら，`multikey-quicksort()`を再帰呼び出しします．このとき，枢軸より小さい区間(`low-m1-1`)と大きい区間(`m2-high`)は`depth`番目の文字でソートを続行します．枢軸と等しい区間(`m1-m2-1`)は`depth+1`番目の文字でソートを行います．もしも，等しい区間の文字(枢軸)がヌル文字であれば，文字列を最後まで比較したので再帰呼び出しは行いません．つまり，同じ文字列が複数個あるということになります．

● 区間の三分割

マルチキー・クイック・ソートのプログラムは，区間を3分割する処理がポイントになります．この処理の良し悪しによって処理時間は大きく左右されますが，幸いなことにBentley氏とSedgewick氏が論文(4)で効率の良い方法を示しています．

図2に示すように，基本的にはクイック・ソートと同様に，左端から枢軸以上のデータを探し，右端から枢軸以下のデータを探して，それを交換することで区間を分割します．このとき，枢軸と等しいデータは一時的に両端へ集めるところがポイントです．つまり，区間を等しいデータ，小さいデータ，大きいデータ，等しいデータの四つに分割するのです．分割が終了したら，等しいデータを中央に集めます．この処理はデータを移動するだけなので簡単です．これで区間を3分割することができます．

● マルチキー・クイック・ソートの実装

それでは，マルチキー・クイック・ソートを用いてブロック・ソートを実装してみましょう．ブロック・ソートの場合，ソートする記号列には終端記号がないので，記号列の大きさで終端を判断します．あとの処理は今までの説明と同じです．プログラムをリスト2に示します．

関数`mquick_sort()`の引き数`low`と`high`が区間，引き数

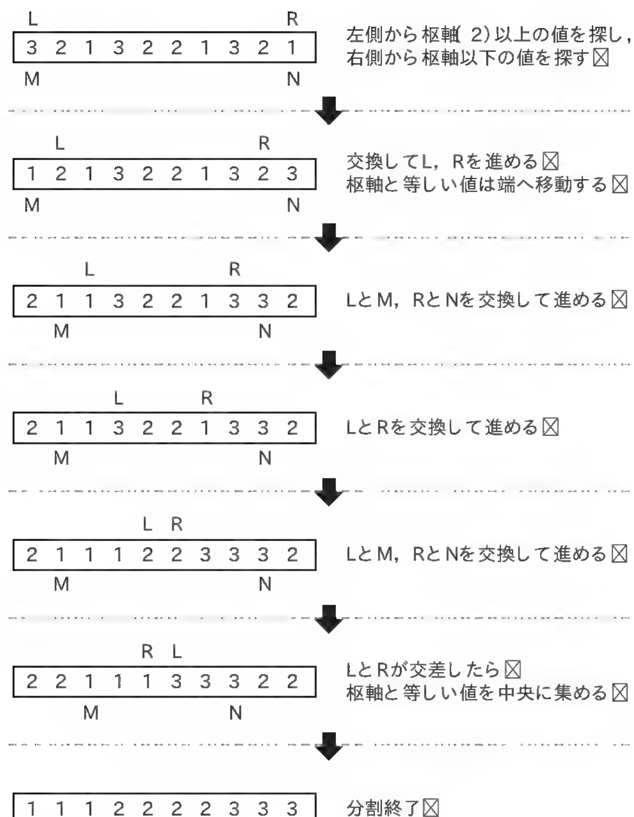


図2　区間の3分割

リスト 2　マルチキー・クイック・ソート

```
typedef unsigned char  Uchar;

/* バッファ定義 */
Uchar buffer[BUFF_SIZE * 2];
Uchar *index_table[BUFF_SIZE];

/* Multikey Quicksort */
void mquick_sort( int low, int high, int depth, int size )
{
  while( 1 ){
    if( high - low + 1 <= ISORT_NUM ){
      insert_sort( low, high, depth, size );
      break;
    } else {
      int pivot, i, j, k, l, m1, m2;
      /* 枢軸の選択 */
      pivot = select_pivot( low, high, depth );
      /* 区間を 4 分割 */
      i = m1 = low;
      j = m2 = high;
      for(;;){
        while( i <= j ){
          k = *(index_table[i] + depth) - pivot;
          if( k > 0 ) break;
          if( k == 0 ){
            /* 枢軸と同じ値を端へ移動 */
            SWAP( i, m1 );
            m1++;
          }
          i++;
        }
        while( i <= j ){
          k = *(index_table[j] + depth) - pivot;
          if( k < 0 ) break;
          if( k == 0 ){
            /* 枢軸と同じ値を端へ移動 */
            SWAP( j, m2 );
            m2--;
          }
          j--;
        }
        if( i > j ) break;
        SWAP( i, j );
        i++;
        j--;
      }
      /* 左端の = を中央に移動 */
      k = MIN( m1 - low, i - m1 );
      for( l = 0; l < k; l++ ) SWAP( low + l, j - l );
      m1 = low + (i - m1);
      /* 右端の = を中央に移動 */
      k = MIN( high - m2, m2 - j );
      for( l = 0; l < k; l++ ) SWAP( i + l, high - l);
      m2 = high - (m2 - j) + 1;
      /* < 部分をソート */
      if( low < m1 ) mquick_sort( low, m1 - 1, depth, size );
      /* > 部分をソート */
      if( m2 <= high ) mquick_sort( m2, high, depth, size );
      /* = 部分をソート */
      if( m1 >= m2 || depth + 1 == size ) break;
      low = m1;
      high = m2 - 1;
      depth++;
    }
  }
}
```

depthが比較する記号の位置，引き数sizeが記号列の大きさを表します．最初のwhileループは，枢軸と等しい区間の再帰呼び出しを繰り返しに変換したものです．区間のデータ数がISORT_NUM（10）よりも少なくなったら，単純挿入ソートinsert_sortに切り替えます．

データ数がISORT_NUMよりも多い場合は区間を3分割します．枢軸は関数select_pivotで選び，変数pivotにセットします．枢軸はlow，(low+high)/2，highの3か所の記号を比較して中央の値を返します．次のforループで区間を4分割したあと，枢軸と等しいデータを中央に集めます．これで区間を3分割することができます．

あとは，枢軸より小さいデータの区間と大きいデータの区間に対してmquick_sort()を再帰呼び出しします．最後に，枢軸と等しい区間をlowとhighにセットし，depthの値を＋1します．そして，whileループの先頭に戻って次の記号でソートを行います．記号列には終端記号がないので，sizeを使って終端をチェックしていることに注意してください．

● マルチキー・クイック・ソートの長所

マルチキー・クイック・ソートが高速なのは，このアルゴリズムに秘密があります．

第1のポイントは，記号列ではなく記号を比較しているところです．当然ですが，記号列よりも記号（整数）の比較のほうが高速です．したがって，記号列を比較して区間を分割するよりも高速に区間を分割することができます．

第2のポイントが，区間を3分割して枢軸と等しい値を集めているところです．これにより，基数ソートと同じような効果を生み出しています．マルチキー・クイック・ソートは，区間を分割するたびに同じ記号を一つの区間に集め，同じ記号の区間であれば次の記号でソートを行います．この動作は記号列の

先頭から順番に基数ソートを行っていることと同じです．これらの効果により，マルチキー・クイック・ソートは記号列を高速にソートすることができるのです．

ブロック・ソートの場合，分布数えソートとマルチキー・クイック・ソートを組み合わせることで，マージ・ソートやクイック・ソートを使うよりも高速にソートすることができます．ところが，これよりも速い方法があるのです．伊東秀夫氏が提案された「二段階ソート法」[5]をブロック・ソートに適用すると，とても高速にソートすることができます．

● 二段階ソート法

二段階ソート法（Tow-Stage Sort）は suffix array を高速に構築する方法で，サフィックス間の関係を利用した画期的なアルゴリズムです．具体的には，i番目の記号S_iとその次（$i+1$番目）の記号S_{i+1}の大小関係を使って，サフィックスをType A とType B の2種類に分けます．

Type A : $S_i > S_{i+1}$

Type B : $S_i \leqq S_{i+1}$

そして，Type B のサフィックスをソートすると，その結果を使ってType A のサフィックスの順序を決定することができます．つまり，Type A のサフィックスはソートする必要がないのです．

簡単な例を示しましょう．図3に示すように，“cacbcccbca”という記号列をブロック・ソートします．ブロック・ソートの場合，ソートするデータはサフィックスではなく記号列になりますが，二段階ソート法のアルゴリズムをそのまま適用することができます．

記号列をType A とType B に分ける処理は，先頭2記号で分布数えソートを行うと簡単です．先頭記号S_1と次の記号S_2を比較して，$S_1 > S_2$であればType A の区間，$S_1 \leqq S_2$であればType B の区間になります．ここで，先頭記号がxの区間を「区間x」，先頭2記号がx，yの区間を「区間xy」と表すことにします．配列index tableは記号列の開始位置（インデックス）を格納します．

図3の区間cを見てください．区間caとcbはType A で，区間ccはType B になります．ここで，区間caの記号列の順序は，区間aの記号列の順序が決まれば，ソートしなくても決定することができます．ここが二段階ソート法を理解するポイントです．区間caはcaで始まる記号列しかありません．これらの記号列の順序は，記号aで始まる記号列の順序関係により決定できるのは明らかです．したがって，区間aの記号列がソートされていれば，その結果を使って区間caの記号列の順序を決めることができるわけです．

この処理は，区間aの先頭から順番にType A の記号列を探して，そのインデックスをindex tableにセットしていくことで実現できます．Type A の記号列は簡単に判別できます．先頭記号aと一つ前の記号（最後尾の記号）xを比較します．もしも$x >$ aであれば，xから始まる記号列はType A であることがわかります．そして，区間xaの先頭から順番にそのインデックスをセットします．区間aのindex tableはソートされているので，見つけた記号列のインデックスを順番にセットしていくだけで，区間xaの記号列を昇順に並べることができるのです．

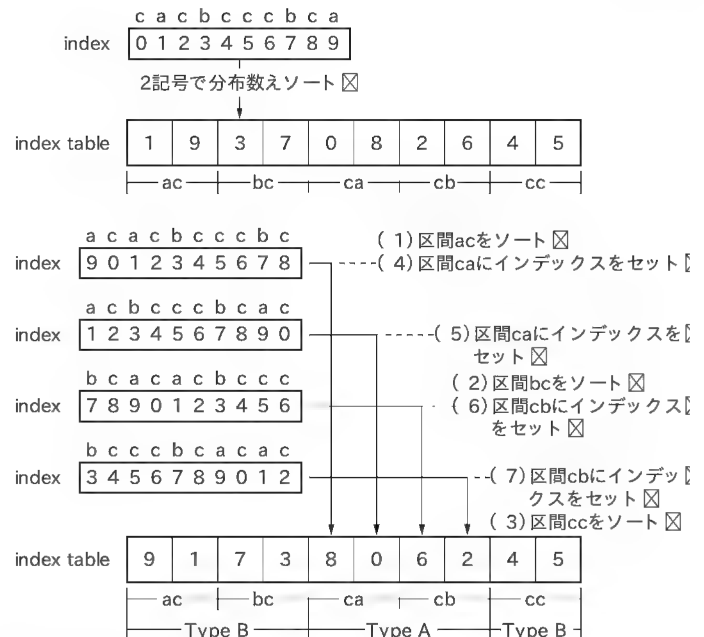


図3　二段階ソート法

具体的には，Type B の区間をすべてソートしたあとで，index tableの先頭から順番にType A の記号列を探して，そのインデックスをセットしていきます．図3の場合，区間aの先頭の記号列はa…cなのでType A です．区間caの先頭にインデックス（8）をセットします．次の記号列もType A なので，インデックス（0）を区間caの2番目にセットします．

このように，index tableからType A の記号列を探して，最後尾の記号xの区間xaにインデックスをセットしていけばいいわけです．この処理は図4のように表すことができます．

記号は{a，b，c，d，e}の5種類とします．この場合，区間のTypeは図4（a）のようになります．記号aはいちばん小さな値なのでType A は存在しません．したがって，記号aの区間はすべてソートすることになります［図4（b）の1］．そして，残りのType B の区間をすべてソートします［図4（b）の2，3，4，5］．

次に，区間aから順番にType A の記号列を探してインデックスをセットしていきます．この処理を「セット処理」と呼ぶことにします．区間aのセット処理で区間ba，ca，da，eaの記号列の順序が決定されます［図4（b）の6］．

区間bの場合，区間baがType A ですが，この区間は区間aのセット処理により，すでに記号列の順序は決定されています．したがって，区間bもすべてソート済みになります．あとはType A の記号列を探してインデックスをセットします．このセット処理で区間cb，db，ebの記号列の順序が決定されます

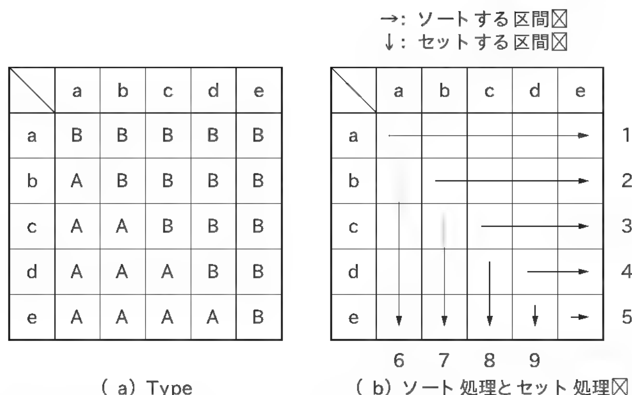


図4　二段階ソート法のソート処理とセット処理

〔図4(b)の7〕.

あとは同様に，区間cのセット処理を行うことで区間dのソートが完了し，区間dのセット処理を行うことができます．

このように，小さい記号から順番にType Aの記号列を探してインデックスをセットしていくことで，Type Aの区間に矛盾なくインデックスをセットすることができるのです．

● 二段階ソート法の実装

二段階ソート法のプログラムをリスト3に示します．

関数sort_two1()を呼び出す前に，先頭の2記号で分布数えソートを行います．このとき，配列count_sumに各記号の累積度数がセットされます．この値から区間の上限値(high)と下限値(low)を求めることができます．変数iが先頭記号，変数jが2番目の記号を表します．すると，i<=jがType Bの区間になるので，この区間だけをマルチキー・クイック・ソートでソートします．

次にType Aのインデックスをセットします．index_tableの先頭から順番にType Aの記号列を探します．*(ptr-1) > *ptrであればType Aの記号列なので，区間*(ptr-1)，*ptrにインデックスptr-1をセットします．区間の先頭は累積度数表count_sumで求めることができます．インデックスをセットしたらcount_sumの値を＋1することで，区間の先頭から順番にインデックスをセットすることができます．

● ratio-2による二段階ソート法の改良

二段階ソート法で実際にソートするのはType Bの記号列だけなので，Type Aの記号列が多くなるほどソートする記号列の個数が少なくなり，高速にソートすることができます．伊東氏の論文(5)によると，Type Bの割合は英文テキスト・ファイルで51%になります．ここで，Type Aの割合を増やすことができれば，もっと高速にソートすることが可能です．伊東氏は論文(5)でType Aの記号列を増やす方法を示しています．

今までの方法は，サフィックスの先頭記号S_1と2番目の記号S_2の関係を利用しています．この方法をratio-1と呼びます．二段階ソート法はratio-1だけではなく，3番目の記号S_3とS_1の関係を利用することができます．つまり，S_1とS_2の関係がType B

リスト3　二段階ソート法(ratio-1)

```
void sort_two1( int size )
{
  int i, j;
  for( i = 0; i < CODE_SIZE; i++ ){
    for( j = i; j < CODE_SIZE; j++ ){
      /* Type B をソート */
      int low  = count_sum[(i << 8) + j];
      int high = count_sum[(i << 8) + j + 1];
      if( high - low > 1 ){
        sort_count += high - low;
        mquick_sort( low, high - 1, 2, size );
      }
    }
  }
  /* Type A をセット */
  for( i = 0; i < size; i++ ){
    Uchar *ptr = index_table[i];
    if( ptr == buffer ) ptr += size;
    if( *(ptr - 1) > *ptr ){
      index_table[ count_sum[(*(ptr - 1) << 8) + *ptr]++ ]
        = ptr - 1;
    }
  }
}
```

のサフィックスでも，$S_1 > S_3$を満たせばType Aのサフィックスとして扱うことができるのです．この方法をratio-2と呼びます．

ratio-2も簡単にプログラムできます．$S_1 \leqq S_2$を満たすType Bの区間で，S_1とS_3を比較して記号列をType AとType Bに分離します．先頭2記号で分布数えソートを行っているので，区間には先頭2記号が同じ記号列しかありません．したがって，S_1とS_3を比較して記号列を簡単に分離することができます．あとは，分離したType Bの記号列だけをソートすればいいのです．

● 二段階ソート法ratio-2の実装

二段階ソート法ratio-2のプログラムをリスト4に示します．

ratio-1のType Bの区間で，ratio-2の関係を使って記号列をType AとType Bに分離します．区間の前半にType Aの記号列を集め，区間の後半がType Bの記号列になります．そして，Type Bの記号列だけをソートします．

Type Aのインデックスをセットする場合，ratio-1とratio-2のチェックが必要になります．ratio-1はリスト3と同じです．ratio-2の場合，*(ptr-2) > *ptrだけではなく，2記号前と1記号前の関係がType Bであること，つまり*(ptr-2) <= *(ptr-1)のチェックも必要になります．

ratio-2の効果はとても高く，伊東氏の論文(5)によるとType Bの割合は英文テキスト・ファイルで35%になります．実際，二段階ソート法をブロック・ソートに適用すると，一般的なテキスト・ファイルであればとても高速にソートすることができます．

マルチキー・クイック・ソートと二段階ソート法の評価結果

それでは，マルチキー・クイック・ソートと二段階ソート法の効果を確かめてみましょう．テスト・データはCanterbury Corpus(9)で配布されているThe Canterbury Corpusの中からalice29.txt，kennedy.xls，plrabn12.txt，ptt5の4ファイルとThe Large Corpusのbible.txtです．プログラ

リスト4 二段階ソート法(ratio-2)

```
void sort_two2( int size )
{
  int i, j, end;
  for( i = 0; i < CODE_SIZE; i++ ){
    for( j = i; j < CODE_SIZE; j++ ){
      int n, low  = count_sum[(i << 8) + j];
      int high = count_sum[(i << 8) + j + 1];
      /* Type B を分離 */
      for( n = low; n < high; n++ ){
        Uchar *ptr = index_table[n];
        if( *ptr > *(ptr + 2) ){
          index_table[n] = index_table[low];
          index_table[low++] = ptr;
        }
      }
      if( high - low > 1 ){
        sort_count += high - low;
        mquick_sort( low, high - 1, 2, size );
      }
    }
  }
  /* Type A をセット */
  for( i = 0; i < size; i++ ){
    /* ratio-1 */
    Uchar *ptr = index_table[i];
    if( ptr == buffer ) ptr += size;
    if( *(ptr - 1) > *ptr ){
      index_table[ count_sum[(*(ptr - 1) << 8) +
                              *ptr]++ ] = ptr - 1;
    }
    /* ratio-2 */
    ptr = index_table[i];
    if( ptr < buffer + 2 ) ptr += size;
    if( *(ptr - 2) > *ptr && *(ptr - 2)
                              <= *(ptr - 1) ){
      index_table[ count_sum[(*(ptr - 2) << 8)
                   + *(ptr - 1)]++ ] = ptr - 2;
    }
  }
}
```

ムはBorland C++5.5.1 for Win32でコンパイルし，筆者のマシン(Windows95, Pentium 166 MHz)で実行しました．結果を表1に示します．時間はファイルの入出力処理を含むプログラム全体の処理時間です．

マージ・ソート(A),(C)とマルチキー・クイック・ソート(B),(D)を比較した場合，マルチキー・クイック・ソートのほうが高速にブロック・ソートすることができます．マルチキー・クイック・ソートはブロック・ソートでも抜群の効果を発揮しています．

二段階ソート法ratio-1(D)，raito-2(E)は，ソートした個数が大幅に減少し，その分だけ高速にブロック・ソートすることができます．特に英文テキスト・ファイルはratio-2の効果がとても高く，処理速度はratio-1よりも高速になります．

ただし，kennedy.xlsは二段階ソート法の効果が少なく，処理速度も英文テキスト・ファイルほど速くなりません．また，ptt5は0が連続しているデータで，マルチキー・クイック・ソートと二段階ソート法を使っても，処理速度はとても遅くなります．このように，二段階ソート法にも弱点があるのです．

なお，これらの結果は筆者のコーディング，実行したマシン，コンパイラなどの環境に大きく依存しています．これらの環境の違いよって，処理時間はかなり左右されることに注意してください．

表1 マルチキー・クイック・ソートと二段階ソート法の評価結果

ファイル名	サイズ	(A)	(B)	(C)	(D)	(E)
alice29.txt	152,089	1.24 (151869)	0.82 (151869)	0.81 (74618)	0.65 (74618)	0.64 (53507)
kennedy.xls	1,029,744	6.76 (1028834)	6.04 (1028834)	5.47 (623977)	4.86 (623977)	4.43 (467927)
plrabn12.txt	481,861	3.68 (481696)	2.58 (481696)	2.59 (234016)	2.09 (234016)	1.99 (162222)
ptt5	513,216	253.6 (512181)	202.0 (512181)	264.6 (477580)	201.1 (477580)	200.5 (465125)
bible.txt	4,047,392	34.37 (4047003)	23.28 (4047003)	21.83 (1943800)	16.57 (1943800)	15.94 (1398313)

(A) 分布数えソートとマージ・ソート
(B) 分布数えソートとマルチキー・クイック・ソート
(C) 二段階ソート法(ratio-1)とマージ・ソート
(D) 二段階ソート法(ratio-1)とマルチキー・クイック・ソート
(E) 二段階ソート法(ratio-2)とマルチキー・クイック・ソート
単位：秒，( )はソートしたデータ数，バッファ 1Mバイト

おわりに

今回はマルチキー・クイック・ソートと二段階ソートによる，速度面での改良を中心に解説しました．後編となる次回は，3分割法による速度のさらなる改善と，0-1-2 codingによる圧縮率の改善法について解説します．

参考文献，URL
(1) 植松友彦；文書データ圧縮アルゴリズム入門，CQ出版社，1994年
(2) 奥村晴彦；C言語による最新アルゴリズム事典，技術評論社，1991年
(3) 奥村晴彦；データ圧縮の基礎から応用まで，C MAGAZINE 2002年7月号，ソフトバンク
(4) Jon Bentley，Robert Sedgewick，*Fast Algorithms for Sorting and Searching Strings*，http://www.cs.princeton.edu/~rs/strings/
(5) 伊東秀夫；Suffix Arrayの効率的な構築法，http://www.ricoh.co.jp/rdc/techreport/No27/
(6) 儘田真吾；suffix arrayの構築――二段階ソート法とその改良，http://www.isl.cs.gunma-u.ac.jp/~shingo/algo.html
(7) 定兼邦彦；大規模テキスト索引(suffix array)の構築法とその情報検索への応用 suffix array構築アルゴリズムと実装，http://www.gi.k.u-tokyo.ac.jp/ssr-homepage/1999/workshop1/
(8) 山本博資；ユニバーサルデータ圧縮アルゴリズムの変遷―基礎から最新手法まで―，http://hirosuke.sr3.t.u-tokyo.ac.jp/files/survey.html
(9) Canterbury Corpus，http://corpus.canterbury.ac.nz/
(10) The bzip2 and libbzip2 home page，http://sources.redhat.com/bzip2/
(11) Zzip，http://debin.org/zzip/
(12) szip homepage，http://www.compressconsult.com/szip/
(13) DO++，http://member.nifty.ne.jp/DO/
(14) white page，http://homepage3.nifty.com/wpage/
(15) M.Hiroi's Home Page，http://www.geocities.co.jp/SiliconValley-Oakland/1680/

ひろい・まこと

TMS320C6713搭載DSPスタータ・キットを使った

C++による DSP オブジェクト指向プログラミング

第8回（最終回） ポリモーフィズムを利用するIIRフィルタ

◆三上 直樹

いよいよこの連載も最後になりました．今回はオブジェクト指向プログラミングの中でも重要な概念の一つであるポリモーフィズム（polymorphism）を利用して，ディジタル・フィルタを作成します．

ディジタル・フィルタには，FIR（Finite-duration Impulse Response）フィルタとIIR（Infinite-duration Impulse Response）フィルタがありますが，FIRフィルタのプログラミングについてはこの連載の第3回目（本誌2004年4月号）で解説したので，今回はIIRフィルタを取り上げます．

1 IIRフィルタ

● IIRフィルタの構成

IIRフィルタとは，そのインパルス応答が無限に続くようなフィルタです．つまり，入力信号が0になっても，0ではない値が無限に出力されるようなフィルタです．ディジタル・フィルタの入出力信号の関係は，差分方程式で表現されます．IIRフィルタでは，入力信号を$x[n]$，出力信号を$y[n]$とすると，次の式で表されます．

$$y[n]=\sum_{m=1}^{M}a_m y[n-m]+\sum_{k=0}^{K}b_k x[n-k] \quad \cdots\cdots(1)$$

この式のa_m（m=1，2，…，M），b_k（k=0，1，…，K）は，フィルタの係数と呼ばれるもので，この値によりフィルタの特性が決定されます．また，nは整数で，時間に対応する値になります．

ところで，この式（1）で，$M=K$としても，いくつかの係数が0であると考えれば，$M \neq K$の場合も含めて取り扱うことができるので，以降では次の式（2）を使うことにします．

$$y[n]=\sum_{m=1}^{M}a_m y[n-m]+\sum_{m=0}^{M}b_m x[n-m] \quad \cdots\cdots(2)$$

フィルタの特性の中でも重要な周波数特性は，フィルタの伝達関数から求めることができます．式（2）から伝達関数$H(z)$を求めると次のようになります注1．

$$H(z)=\frac{\sum_{m=0}^{M}b_m z^{-m}}{1-\sum_{m=1}^{M}a_m z^{-m}} \quad \cdots\cdots(3)$$

式（3）の分子と分母は，ともにz^{-1}に関してM次の多項式になっているため，このような伝達関数で表されるフィルタはM次のフィルタと呼ばれます．この式（3）で，$z=\exp(j\omega T)$注2と置き換えると，周波数特性を表す関数である周波数応答（frequency response）が得られます．

IIRフィルタのプログラムを作成するためには，基本的に式（2）を実現するようなプログラムを作成すればよいことになります．また，この式を変形することで，いろいろな構成方法を導くことができます．以下では，今回プログラムを作成する際に利用する直接形IIと縦続形という二つの構成方法について簡単に説明します．

● 直接形II

式（2）から導かれるIIRフィルタの構成は，直接形（direct form）と呼ばれています．しかし，そのほかにも直接形と呼ばれるいくつかの異なった構成があります．そこで，これらを区別するため，式（2）に一対一に対応しているものは直接形Iと呼ばれています．

一方，以下でプログラムを作成する際には，図1に示すブ

注1：$x[n]$のz変換を$X(z)$，$y[n]$のz変換を$Y(z)$とし，初期条件を0とすると，伝達関数$H(z)$は次のように定義される．

$$H(z)=Y(z)/X(z)$$

一方，z変換は線形な変換であり，$f[n]$のz変換を$F(z)$とすると，次の関係が成り立つ．

$f[n-1]$のz変換は$F(z)z^{-1}$，$f[n-2]$のz変換は$F(z)z^{-2}$，…

したがって，式（2）の両辺をz変換すると次のようになる．

$$Y(z)=\sum_{m=1}^{M}a_m Y(z)z^{-m}+\sum_{m=0}^{M}b_m X(z)z^{-m}$$

この式から式（3）を導くことができる．

注2：jは虚数単位，ωは角周波数，Tは標本化間隔を示す．

ロック図で表されるような構成を使うことにします．この構成は直接形IIと呼ばれています．この場合，入出力の関係を表す差分方程式は式(1)とは異なり，式(4)のようになります．

$$\begin{cases} u[n]=\sum_{m=1}^{M} a_m u[n-m]+x[n] \\ y[n]=\sum_{m=0}^{M} b_m u[n-m] \end{cases} \quad \cdots\cdots(4)$$

● 縦続形

式(3)の伝達関数でMが偶数の場合，分子，分母はともに2次の多項式の積という形で書くことができます．$N=M/2$と置くと，次のようになります[注3]．

$$H(z)=\frac{b_{01}+b_{11}z^{-1}+b_{21}z^{-2}}{1-a_{11}z^{-1}-a_{21}z^{-2}}\times\frac{b_{02}+b_{12}z^{-1}+b_{22}z^{-2}}{1-a_{12}z^{-1}-a_{22}z^{-2}}\times\cdots\times\frac{b_{0N}+b_{1N}z^{-1}+b_{2N}z^{-2}}{1-a_{1N}z^{-1}-a_{2N}z^{-2}}$$
$$=H_1(z)\times H_2(z)\times\cdots\times H_N(z) \quad \cdots\cdots(5)$$

この式に対応するフィルタが，縦続形(cascade form)のIIRフィルタです．このブロック図は，図2のようになります．また，入出力の関係を表す差分方程式は次のようになります．

$$\begin{cases} u_1[n]=a_{11}u_1[n-1]+a_{21}u_1[n-2]+x[n] \\ y_1[n]=b_{01}u_1[n]+b_{11}u_1[n-1]+b_{21}u_1[n-2] \\ u_2[n]=a_{12}u_2[n-1]+a_{22}u_2[n-2]+y_1[n] \\ y_2[n]=b_{02}u_2[n]+b_{12}u_2[n-1]+b_{22}u_2[n-2] \\ \quad\vdots \\ u_N[n]=a_{1N}u_N[n-1]+a_{2N}u_N[n-2]+y_{N-1}[n] \\ y[n]=b_{0N}u_N[n]+b_{1N}u_N[n-1]+b_{2N}u_N[n-2] \end{cases} \quad \cdots\cdots(6)$$

2 直接形IIフィルタのプログラム

直接形IIのIIRフィルタのクラスは，第7回目(本誌2004年9月号)にも作りましたが，そこではこのフィルタのクラスについて詳しく説明しなかったので，ここで説明を加えておきます．なお，ここで作るクラスは第7回目のものを拡張したものです．

● 直接形II用のクラス

直接形IIRフィルタは，縦続形IIRフィルタに比べて係数の誤差や演算誤差の影響を受けやすいという性質をもっています．特に，フィルタの遮断周波数が標本化周波数に比べて非常に低い場合や，標本化周波数の1/2に非常に近い場合に，これらの

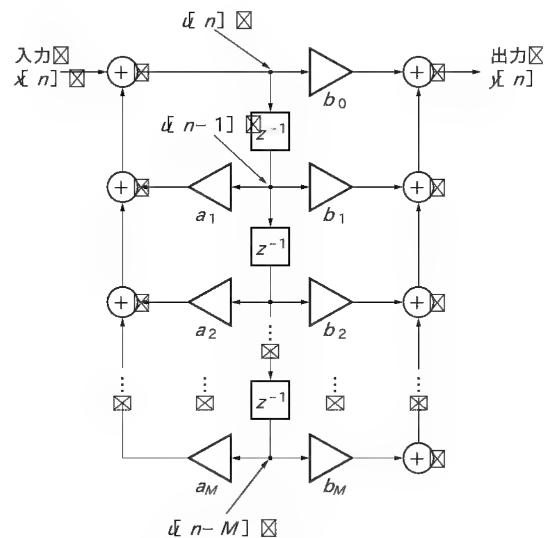


図1　直接形IIのIIRフィルタの構成

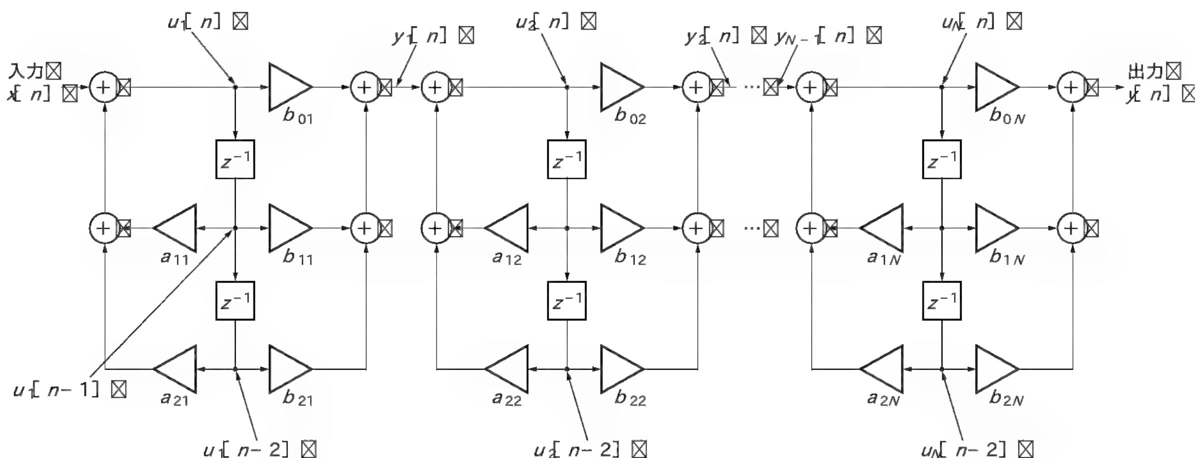


図2　縦続形IIRフィルタの構成

注3：Mが奇数の場合は，$N=(M+1)/2$とする．さらに$H_1(z)$，$H_2(z)$，…，$H_N(z)$の中の一つを$H_k(z)$とすると，$a_{2k}=0$，$b_{2k}=0$，つまり，$H_k(z)=(b_{0k}+b_{1k}z^{-1})/(1-a_{1k}z^{-1})$とすればよい．

リスト 1　直接形IIのIIRフィルタ実現のためのテンプレート・クラス（classIIR_Direct2.cpp）

```
//------------------------------------------------------------
// 直接形IIのIIRフィルタ用テンプレート・クラス
//------------------------------------------------------------

#ifndef MK_My_IIRDirect2

//------------------------------------------------------
// IIRフィルタ用クラス，直接形II
//------------------------------------------------------
template <class T, int nOrd> class IIR_Direct2
{
private:
    T un[nOrd+1];
    const T *const ak, *const bk;
public:
    IIR_Direct2(const T am[], const T bm[])
            : ak(am), bk(bm)
    { for (int k=0; k<=nOrd; k++) un[k] = 0.0; }
    inline float Execute(const float xin);
};

template <class T, int nOrd>
inline float IIR_Direct2<T, nOrd>::Execute(const float xin)
{
    T utmp = xin;
    for (int m=0; m<nOrd; m++) utmp = utmp + ak[m]*un[m+1];
    un[0] = utmp;
    T yn = 0.0;
    for (int m=0; m<=nOrd; m++) yn = yn + bk[m]*un[m];
    for (int m=nOrd; m>0; m--) un[m] = un[m-1];

    return yn;
}

#define MK_My_IIRDirect2
#endif
```

誤差の影響が大きく現れます．また，フィルタの次数が高くなればなるほど，これらの誤差の影響を大きく受けるようになります．

係数に誤差があると，周波数特性が本来のものから変化してしまいます．また，演算誤差があると，雑音が発生します．さらに，いずれの誤差もその程度が大きくなると，フィルタが発振してしまう場合もあります．

しかし，係数やデータの語長を十分長くすれば，直接形でも使うことが可能になります．いろいろと試してみたところ，係数の型や演算の際のデータ型は，float型でも十分なこともあれば，double型が必要な場合もありました．そこで，両方の型を使えるようにするため，直接形IIRフィルタのクラスをtemplateクラスとして作成します．

直接形IIRフィルタのクラスIIR_Direct2をリスト1（classIIR_Direct2.cpp）に示します．このクラスの宣言は，

```
template <class T, int nOrd>
                                class IIR_Direct2
```

から始まり，テンプレート・クラスになっています．

このクラスは二つのテンプレート引き数を持ちます．最初のテンプレート引き数は，classというキーワードに続いて，クラスの中で使うデータ型を表す文字列（この例ではT）という形式になっています．この書式は，テンプレート引き数の書き方としておなじみのものです．2番目のテンプレート引き数は，classではなくintのような型名を書き，続いてnOrdのような名前を書くという形式になっています．このような形式のテンプレート引き数は，定数式引き数（constant expression parameters）といわれます．リスト1の例では，nOrdはクラス内部の配列のサイズなどを指定するために使われおり，この値はフィルタの次数に対応します．

このクラスのオブジェクトを宣言するときの例を示します．オブジェクトの名前をLPF1とし，データ型をfloat型，次数をorderとすると，次のように記述します．

```
IIR_Direct2<float, order> LPF1(arg1, arg2);
```

このとき注意すべき点は，orderは値がすでに定義されている定数でなければならないということです．これを変数にすると，コンパイル・エラーになります．その理由は，クラス内部のプログラムでorderを使っている部分は，コンパイル時に決定されるからです．

▶ 非公開部

配列unは，式（4）の$u[n]$に相当するものです．このデータ語長は，演算誤差と密接な関係があるのでT型とし，オブジェクトを宣言するときにこの型を指定できるようにしています．

akとbkはフィルタの係数を指すポインタで，このポインタもT型として宣言します．この宣言では，constが2か所にあることに注意してください．最初のconstはポインタの指す内容です．この場合はフィルタの係数になりますが，これがconstであるということです．2番目の*constは，ポインタがconstであるということです．

▶ 公開部

コンストラクタでは，ポインタak，bkをメンバ初期設定の機能を使って設定し，配列unの内容を0に初期化します．

メンバ関数Execute()は，直接形IIRフィルタを実行するためのものです．この関数のローカル変数であるynとutmp[注4]は，演算誤差と密接な関係があるので，T型の変数として宣言しています．

● 直接形II用のクラスを使ったIIRフィルタの実現

直接形IIRフィルタ用のクラスIIR_Direct2を使ってIIRフィルタを実現した例をリスト2（IIR_Direct.cpp）に示します．ここでは連立チェビシェフ特性で5次のフィルタを実現します．フィルタを設計したときのパラメータを表1に示します．また，その振幅特性を図3に示します．

実現するフィルタは，標本化周波数（48kHz）に対して遮断周波数（2kHz）がかなり低くなっています．このような場合は，演算誤差の影響が大きく現れる可能性があります．そこで，クラ

注4：これらの変数をメンバ関数内のローカル変数ではなく，クラスのデータ・メンバとして宣言することもできるが，メンバ関数内のローカル変数にしたほうが実行効率が良くなる．

リスト 2　直接形IIのIIRフィルタ用テンプレート・クラスを使って実現したIIRフィルタ(IIR_Direct.cpp)

```
//--------------------------------------------------------------
//  直接形IIのIIR低域通過フィルタ
//      連立チェビシェフ特性のフィルタ(楕円フィルタ)
//      次数            7
//      標本化周波数    48.000000 kHz
//      遮断周波数      2.000000 kHz
//      通過域のリプル  0.20 dB
//      阻止域のリプル  40.00 dB
//--------------------------------------------------------------
#include "AIC23_Polling.hpp"
#include "classIIR_Direct2.cpp"

const int ORDER = 7;
const double amD[ORDER] =
    {    6.4824931281E+00, -1.8158778276E+01,  2.8482078515E+01,
        -2.7008235419E+01,  1.5479818828E+01, -4.9646327290E+00,
         6.8723598287E-01};
const double bmD[ORDER+1] =
    {    5.8004454915E-03, -2.6712907186E-02,  4.5606643993E-02,
        -2.4684197588E-02, -2.4684197588E-02,  4.5606643993E-02,
        -2.6712907186E-02,  5.8004454915E-03};
float amF[ORDER], bmF[ORDER+1];

int main()
{
    float chIN[2], chOUT[2];
    AIC23_Polling codec;
    IIR_Direct2<double, ORDER> dblLPF(amD, bmD);
                                            // double型のオブジェクト
    IIR_Direct2<float, ORDER>  fltLPF(amF, bmF);
                                            // float型のオブジェクト

// 係数の型変換:  double型 → float型
    for (int i=0; i<ORDER; i++) amF[i] = amD[i];
    for (int i=0; i<ORDER+1; i++) bmF[i] = bmD[i];

    while(1)
    {
        codec.Read(chIN);
        chOUT[0] = dblLPF.Execute(chIN[0]);
                                    // double型で実行するフィルタ
        chOUT[1] = fltLPF.Execute(chIN[0]);
                                    // float型で実行するフィルタ
        codec.Write(chOUT);
    }
}
```

ス IIR_Direct2 がオブジェクトを宣言する際に，テンプレート引き数を double 型にしたものと，float 型にしたものを用意し，両者の比較が行えるようにしました．

フィルタ処理は CH0 の入力信号に対してのみ行い，double 型で実行したフィルタの出力信号を CH0 に，float 型で実行したフィルタの出力信号を CH1 にそれぞれ出力します．

● 実行結果

リスト 2のフィルタに 1.8kHz の正弦波を入力したしたときの出力の波形を写真 1に示します．上の波形は double 型のオブジェクトを使った場合，下の波形は float 型のオブジェクトを使った場合です．

この写真から，double 型の場合はきれいな正弦波が出力されていることがわかります．一方，float 型の場合は波形が上下に揺らいでいるようすがわかります．これは演算誤差の影響で雑音が発生したことが原因です．さらに，float 型の場合は double 型に比べて振幅が大きくなっていますが，これはフィルタの振幅特性が本来のものから変化したことによるものです．

そこで，係数を double 型にした場合と float 型にした場

表 1　低域通過 IIR フィルタの設計時に与えたパラメータ

標本化周波数	48kHz
通過域端の周波数	2.0kHz
次数	7
通過域のリプル	0.2dB
阻止域の減衰量	40dB
振幅特性の形状	連立チェビシェフ型

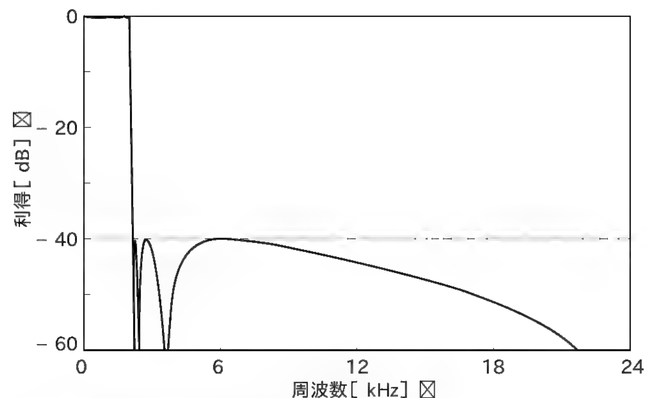


図 3　作成する低域通過フィルタの振幅特性

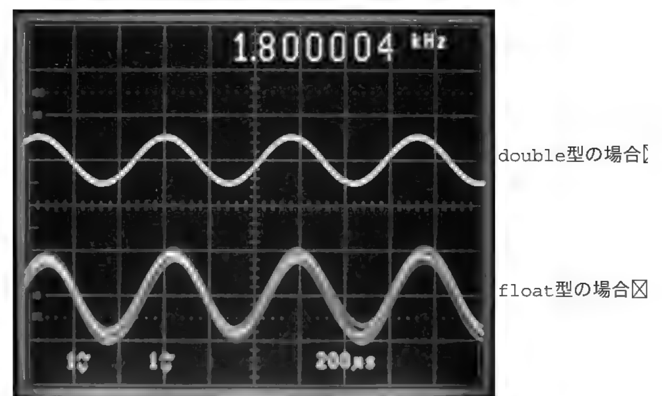


写真 1　直接形IIの IIR フィルタの出力波形

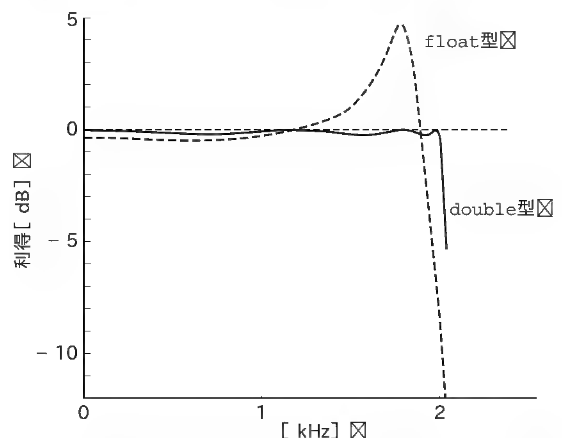
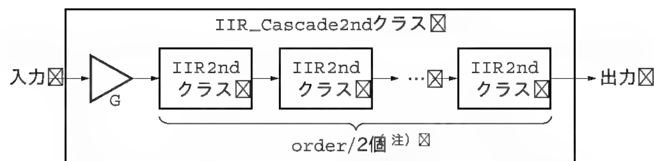


図 4　リスト 2のフィルタについて，その係数が float 型および double 型の場合の振幅特性(通過域を拡大)



(注)order：次数，ただしorderが奇数の場合は(order＋1)/2個

図5　クラスによる縦続形IIRフィルタのイメージ

合の振幅特性の変化を示します．ここで作成したフィルタの場合，阻止域の特性は設計値とそれほど異なりません．一方，係数がfloat型の場合に，通過域では設計値と大きく異なっています．そこで，図4(p.191)には通過域付近を拡大したものを示します．係数をfloat型にした場合は，1.8kHz付近で5dB程度のピークが生じており，この周波数付近で振幅特性は大きな誤差を持つことがわかります．写真1で，係数がfloat型の場合，振幅がdouble型の場合に比べて大きくなっているのは，このことが原因です．

以上のことから，直接形の場合，float型で演算を行う場合や，係数をfloat型にする場合には，精度が不足する場合があることがわかります．

3　縦続形フィルタのプログラム

直接形のIIRフィルタは，前項で示したように誤差の影響を大きく受けるという欠点があるため，実際にはあまり使われません．よく使われるのは，縦続形のIIRフィルタです．縦続形は図2に示すように，2次の直接形IIRフィルタを一つの基本単位とし，この基本単位を縦続接続した構成になります．この基本単位には，すでに作成したクラスIIR_Direct2を使うこともできます．

しかし，ここでは縦続形構成の基本単位である2次のIIRフィルタに対応するクラスIIR2ndを新たに作成し，このクラスのオブジェクトを使った縦続形IIRフィルタのためのクラスIIR_Cascade2ndを作成します．つまり，クラスIIR_Cascade2ndは，クラスIIR2ndを包含するようにします[注5]．したがって，イメージ的には図5のようになります．リスト3(IIR_Cascade2nd.cpp)にこのプログラムを示します．

なお縦続形は，直接形に比べて誤差の影響はそれほど大きくは現れません．したがって，リスト3のプログラムでは，フィルタ係数およびフィルタを実行するための計算で使う変数はfloat型にしています．

● 縦続形IIRフィルタの基本単位となるクラス IIR2nd

▶ 非公開部

フィルタの係数に対応するポインタak，bkと，フィルタ処理で使う変数un1，un2をメンバとして宣言しています．

▶ 公開部

コンストラクタの宣言では，二つの引き数am[]およびbm[]

リスト3　2次のIIRフィルタを基本単位とする縦続形のIIRフィルタで利用するテンプレート・クラス(IIR_Cascade2nd.cpp)

```
//-------------------------------------------------------------
// 縦続形IIRフィルタ用テンプレート･クラス，
//       2次の基本単位(IIR2nd)を静的配列の要素とするもの
//-------------------------------------------------------------
#ifndef MK_My_IIRCascade2nd

//-------------------------------------------------------------
// IIRフィルタの基本単位(2次)
//-------------------------------------------------------------
class IIR2nd
{
private:
    const float *ak, *bk;
    float un1, un2;
public:
    IIR2nd(const float am[] = NULL, const float bm[] = NULL)
        : ak(am), bk(bm) { un1 = un2 = 0.0; }
    inline float Execute(const float xin);
};

inline float IIR2nd::Execute(const float xin)
{
    float un = xin + ak[0]*un1 + ak[1]*un2;
    float yn = bk[0]*un + bk[1]*un1 + bk[2]*un2;
    un2 = un1;
    un1 = un;

    return yn;
}

//-------------------------------------------------------------
// 2次の基本単位(IIR2nd)を静的配列の要素とする縦続形IIRフィルタ
//-------------------------------------------------------------
template <int nOrd> class IIR_Cascade2nd
{
private:
    IIR2nd section[(nOrd+1)/2];
    const int nOrd2;         // (order+1)/2, order: 次数
    const float G;           // 利得定数
public:
    struct Coefs{ float am[2], bm[3]; };
    IIR_Cascade2nd(const Coefs ajbj[], const float A);
    inline float Execute(const float xin);
};

// 縦続形構成のためのコンストラクタ
template <int nOrd>
IIR_Cascade2nd<nOrd>::IIR_Cascade2nd(const Coefs ab[],
                                     const float A)
                     : nOrd2((nOrd+1)/2), G(A)
{
    for (int n=0; n<nOrd2; n++)
        section[n] = IIR2nd(ab[n].am, ab[n].bm);
}

// フィルタ処理を実行するためのメンバ関数
template <int nOrd>
inline float IIR_Cascade2nd<nOrd>::Execute(const float xin)
{
    float ym = G*xin;
    for (int m=0; m<nOrd2; m++) ym = section[m].Execute(ym);

    return ym;
}

#define MK_My_IIRCascade2nd
#endif
```

注5：このような関係はhas-a関係と呼ばれる．

注6：このほかに，引き数を持たないコンストラクタもデフォルト・コンストラクタである．

にデフォルトとしてNULLが指定されていることに注意してください．このようにデフォルトの引き数が与えられているコンストラクタはデフォルト・コンストラクタ(default constructor)[注6]と呼ばれています．デフォルト・コンストラクタを定義したのは，このクラスを包含するクラスIIR_Cascade2ndのコンストラクタで行っているような初期化の操作，つまりIIR2ndオブジェクトをIIR2ndオブジェクトの配列に代入するという操作を可能にするためのものです．このようなデフォルト・コンストラクタが定義されていない場合，初期化のために代入を行うという記述はコンパイル・エラーになります．

Execute()は，フィルタ処理を実行するためのメンバ関数です．

● 縦続形IIRフィルタの全体に対応するクラス **IIR_Cascade2nd**

▶ 非公開部

縦続形の基本単位となる2次の直接形IIRフィルタに対応するクラスIIR2ndのオブジェクトを配列として宣言しています．その配列のサイズは，このクラスのテンプレート引き数により決定されます．

そのほか，クラスの内部で使う定数を宣言しています．

▶ 公開部

構造体として宣言されているCoefsは，縦続形の基本単位1個の係数に対応します．この構造体の内部は二つの配列からなり，am[]は式(5)の分母の係数に，bm[]は式(5)の分子の係数に対応します．対応関係は次のようになります．

a_{1m}： am[0]，a_{2m}： am[1]

b_{0m}： bm[0]，b_{1m}： bm[1]，b_{2m}： bm[2]

コンストラクタは，メンバ初期設定の機能を使って，nOrd，Gの値を設定します．さらに，クラスIIR2ndの配列として宣言されているオブジェクトの初期化を行います．このとき，forループで，

section[n] = IIR2nd(ab[n].am, ab[n].bm);

という代入文を実行していますが，このように記述できるのは，クラスIIR2ndの中でデフォルト・コンストラクタが定義されているからです．クラスIIR2ndの中でデフォルト・コンストラクタが定義されていない場合には，ここでコンパイル・エラーが発生します．

メンバ関数Execute()は，フィルタ処理を実行するためのものです．

● 縦続形のクラスを使ったIIRフィルタの実現

リスト4(Cascade1.cpp)にクラスIIR_Cascade2ndを使って作成したIIRフィルタのプログラムを示します．フィルタの振幅特性はリスト2と同じものです．網掛けを行った部分は，次項で作成するプログラムで置き換わる部分です．

係数は構造体Coefsに従って宣言します．この構造体はクラスIIR_Cascade2ndの内部で宣言されているので，Coefsの先頭にIIR_Cascade2nd<ORDER>::を付加します．

縦続形フィルタの次数が奇数の場合，縦続形の基本単位の中

リスト4 クラスIIR_Cascade2ndを使って実現した縦続形IIRフィルタ(Cascade1.cpp)

```
//----------------------------------------------------------------
//  縦続形のIIR低域通過フィルタ(クラスIIR_Cascade2ndを使用)
//      連立チェビシェフ特性のフィルタ(楕円フィルタ)
//      次数            7
//      標本化周波数    48.000000 kHz
//      遮断周波数      2.000000 kHz
//      通過域のリプル  0.20 dB
//      阻止域のリプル  40.00 dB
//----------------------------------------------------------------
#include "AIC23_Polling.hpp"
#include "IIR_Cascade2nd.cpp"

const int ORDER = 7;
const int ORDER2 = (ORDER+1)/2;

const IIR_Cascade2nd<ORDER>::Coefs anbn[ORDER2] =
    {{{  0.8709321289,             0},    {    1.0,  1.0,                   0}},
     {{  1.8078614303, -0.8448719342},    {    1.0, -1.7851823868,        1.0}},
     {{  1.8848204731, -0.9452135174},    {    1.0, -1.9014398642,        1.0}},
     {{  1.9188790958, -0.9880997086},    {    1.0, -1.9186976722,        1.0}}};
const float g = 5.8004454915E-03;

IIR_Cascade2nd<ORDER> lpf(anbn, g);

int main()
{
    float ch[2];
    AIC23_Polling codec;

    while(1)
    {
        codec.Read(ch);
        ch[0] = lpf.Execute(ch[0]);
        codec.Write(ch);
    }
}
```

リスト5　1次および2次のIIRフィルタを基本単位とする縦続形のIIRフィルタで利用するテンプレート・クラス（IIR_Cascade1st2nd.cpp）

```
//-----------------------------------------------------------
// 縦続形のIIRフィルタのためのクラス
//     BaseIIR1st2nd, biLinear, biQuad, IIR_Cascade1st2nd
//-----------------------------------------------------------
#ifndef MK_My_Cascade1st2nd

//-----------------------------------------------------------
// 1次および2次の基本単位のための抽象基底クラス
//-----------------------------------------------------------
class BaseIIR1st2nd
{
protected:
    const float *const ak, *const bk;
public:
    BaseIIR1st2nd(const float am[], const float bm[])
            : ak(am), bk(bm) {}
    inline virtual ~BaseIIR1st2nd() = 0;
    inline virtual float Execute(const float xin) = 0;
};

// 何も行わないデストラクタの定義
inline BaseIIR1st2nd::~BaseIIR1st2nd() {}
//-----------------------------------------------------------

//-----------------------------------------------------------
// IIRフィルタの基本単位(1次)に対応するクラス
//-----------------------------------------------------------
class biLinear : public BaseIIR1st2nd
{
private:
    float un1;
public:
    biLinear(const float am[], const float bm[])
            : BaseIIR1st2nd(am, bm) { un1 = 0.0; }
    inline float Execute(const float xin);
};

inline float biLinear::Execute(const float xin)
{
    float un = xin + ak[0]*un1;
    float yn = bk[0]*un + bk[1]*un1;
    un1 = un;

    return yn;
}
//-----------------------------------------------------------

//-----------------------------------------------------------
// IIRフィルタの基本単位(2次)に対応するクラス
//-----------------------------------------------------------
class biQuad : public BaseIIR1st2nd
{
private:
    float un1, un2;
public:
    biQuad(const float am[], const float bm[])
            : BaseIIR1st2nd(am, bm) { un1 = un2 = 0.0; }
    inline float Execute(const float xin);
};

inline float biQuad::Execute(const float xin)
{
    float un = xin + ak[0]*un1 + ak[1]*un2;
    float yn = bk[0]*un + bk[1]*un1 + bk[2]*un2;
    un2 = un1;
    un1 = un;

    return yn;
}
//-----------------------------------------------------------

//-----------------------------------------------------------
// 縦続形IIRフィルタのためのクラス
//-----------------------------------------------------------
template<int order> class IIR_Cascade1st2nd
{
private:
    BaseIIR1st2nd *Hm[(order+1)/2];
    const int   nSection;        // 基本単位の数
    const int   nOrder;          // 次数
    const float Gain;
public:
    struct Coefs{ float am[2], bm[3]; };
    IIR_Cascade1st2nd(const Coefs ab[], const float g);
    ~IIR_Cascade1st2nd();
    inline float Execute(const float xin);
};

template<int order>
IIR_Cascade1st2nd<order>::IIR_Cascade1st2nd(const Coefs ab[],
                                             const float g)
    : nSection((order+1)/2), nOrder(order), Gain(g)
{
    if ( (nOrder & 0x1) == 0 )
    {                                // 偶数次の場合
        for (int m=0; m<nSection; m++)
            Hm[m] = new biQuad(ab[m].am, ab[m].bm);
    }
    else
    {                                // 奇数次の場合
        Hm[0] = new biLinear(ab[0].am, ab[0].bm);
        for (int m=1; m<nSection; m++)
            Hm[m] = new biQuad(ab[m].am, ab[m].bm);
    }
}

template<int order>
IIR_Cascade1st2nd<order>::~IIR_Cascade1st2nd()
{
    for (int m=0; m<nSection; m++) delete Hm[m];
}

template<int order>
inline float IIR_Cascade1st2nd<order>::Execute(const float xin)
{
    float ym = Gain*xin;
    for (int m=0; m<nSection; m++) ym = Hm[m]->Execute(ym);

    return ym;
}
//-----------------------------------------------------------

#define MK_My_Cascade1st2nd
#endif
```

で，一つの基本単位は1次のフィルタになります．ここでは先頭の基本単位を1次のフィルタにします．そのため，係数を宣言する場合は a_{21} と b_{21} に対応するところを0にしています．

フィルタ処理はCH0の信号に対してのみ行っています．

実行した結果は前項の写真1の上の波形と同じになるので，省略します．

4 ポリモーフィズムを利用する縦続形フィルタ

縦続形IIRフィルタの基本単位は前にも説明したとおり，1次および2次のIIRフィルタです．しかし，前項では基本単位をすべて2次のIIRフィルタ用のクラスを使って縦続形のIIRフィルタを作っています．

そこで，ここでは基本単位として1次および2次のIIRフィルタに対応する二つのクラスを作成し，それを基にポリモーフィズム（polymorphism）を利用して縦続形のフィルタを実現します[注7]．

● 縦続形IIRフィルタ用のクラス

リスト5（IIR_Cascade1st2nd.cpp）に縦続形IIRフィルタを実現するためのクラスを示します．縦続形IIRフィルタを実現するためには，図2から，1段目の出力を2段目に入力し，2段目の入力を3段目へ入力し…という処理を行えばよいことがわかります．ここでは，この処理を単純なfor文だけ，つまりfor文の中にif文などの条件文を入れずに実現することを考えます．

そこで，まず縦続形IIRフィルタの基本単位に対応する基底クラスとなるクラスBaseIIR1st2ndを作成します．次に，この基底クラスを派生する形で1次IIRフィルタのためのクラスbiLinear，および2次のIIRフィルタのためのクラスbiQuadを作成します．この関係を図6に示します．

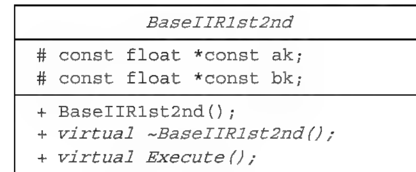
このように，継承を利用して縦続形IIRフィルタの基本単位に対応するクラスを作っておくと，単純なfor文だけで縦続形IIRフィルタを実行する部分のプログラムを作成することができます．そのようにして作成した縦続形IIRフィルタを実現するクラスがIIR_Cascade1st2ndです．このイメージを図7に示します．

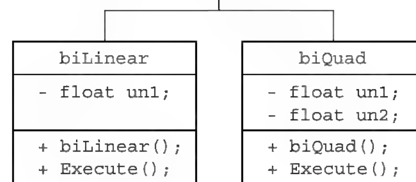
▶ 基本単位の基底クラス：**BaseIIR1st2nd**

BaseIIR1st2ndは，縦続形IIRフィルタの基本単位に対応する1次および2次のIIRフィルタの基底クラスです．

限定公開部では，フィルタの係数に対応するポインタを宣言しています．

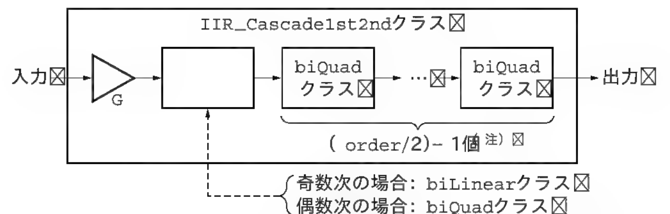
公開部では，コンストラクタ，デストラクタ，およびフィルタを実行するためメンバ関数Execute()を宣言しています．





+: 公開メンバ
#: 限定公開メンバ
-: 非公開メンバ
（斜体は抽象クラスおよび純粋仮想関数を表す）

図6　クラスの継承の関係
縦続形IIRフィルタに対応するテンプレート・クラスIIR_Cascade1st2ndを構成する，1次および2次のIIRフィルタに対応する，基本単位として使うクラスの継承の関係を示す



（注）order：次数，ただしorderが奇数の場合は（（order+1）/2）-1個

図7　ポリモーフィズムを利用した縦続形IIRフィルタのイメージ

デストラクタとメンバ関数Execute()は，純粋仮想関数にしています．

デストラクタを純粋仮想関数にしているのは，第7回目のコラム3（本誌2004年9月号のp.159）で説明しているので，そちらを参照してください．ただし，今回作成するプログラム中で使う場合は，仮想デストラクタは必要ありません．しかし，このクラスを継承したクラスがデストラクタを持つ場合には仮想デストラクタが必要となるため，念のために宣言しておきました．

純粋仮想デストラクタを宣言した場合，通常はこれをオーバライドするデストラクタを派生クラスの中で定義します．しかし，ここで作成する派生クラスはデストラクタを持つ必要がありません．その場合は，リスト5のように基底クラスのデストラクタとして定義してもさしつかえはありません．なお，このデストラクタは何も処理を行わないデストラクタです．

▶ 基本単位の派生クラス：**biLinear，biQuad**

クラスbiLinearおよびbiQuadはBaseIIR1st2ndの派生クラスです．それぞれ，コンストラクタとフィルタを実行す

注7：もちろん，ポリモーフィズムを使わなくてもプログラムを記述することはできる．しかし，ポリモーフィズムを使ったほうが，if文など，処理の流れが分岐する箇所を減らすことができ，プログラムの構造が単純になる場合が多い．その結果，プログラムのデバッグや保守管理が楽になる．

Column 実行時間の測定方法

実行時間を測定するにはいくつかの方法があり，Code Composer Studio(CCS)もそのための機能を提供しています．しかし，CCSが持つ実行時間を測定する機能を使う方法は，慣れないと設定に手間がかかるため，ここではもっとも単純な方法を紹介します．その方法とは，処理の前後に外部に対してパルス信号を出力し，それをオシロスコープで測定するという方法です．

この連載で使っているDSPボードから信号を外部に取り出す場合には，タイマの出力を利用すると簡単です．タイマ出力はドータ・ボード用のコネクタ(J3, Peripheral Expansion Connector)の49ピンから取り出すことができます．

タイマを使う場合，CSL(Chip Support Library)を使うと便利です．そこで，これを使って実行時間測定用のパルスを発生するためのプログラムをリストAに示します．タイマとしてはチャネル1のタイマを使います．パルスを発生するためのクラスがExecMeasureで，パルスを発生するメンバ関数はExecute1()とExecute2()です．二つ用意した理由は，パルスの幅を変えて二つのパルスを区別できるようにするためです．Execute1()は，Execute2()の3倍の幅のパルスを発生します．

このプログラムでは，クラスExecMeasureのオブジェクトTOUTも宣言しています．したがって，このプログラムを使う場合は，まずExecMeasure.cppをインクルードし，測定箇所の前にTOUT.Execute1()を置き，測定箇所の後にTOUT.Execute2()を置くだけです．

たとえば，リスト4のプログラムでフィルタの実行時間を測定する場合は，次のようになります．

```
    ⋮
TOUT.Execute1();
ch[0] = lpf.Execute(ch[0]);
TOUT.Execute2();
    ⋮
```

実行時間測定のためにオシロスコープのプローブを接続しているようすを写真A.1に示します．49ピンにプローブ先端を，79ピンにアース・リードを接続しています．写真A.2にはオシロスコープに二つのパルスが表示されているようすを示します．実行時間は731.0nsと表示されています．

参考までに，このような方法で今回作成したIIRフィルタの実行時間を測定した結果を，表Aに示します．

リストA　実行時間測定のためのパルスを発生するクラス(ExecMeasure.cpp)

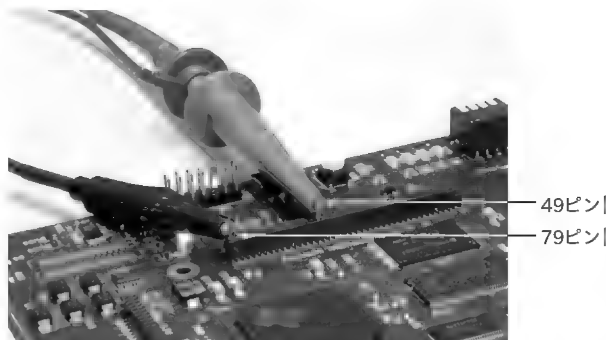
```
//----------------------------------------------------------
// 実行時間測定のためのパルス発生器
//       出力: timer1 の TOUT1 ピン
//----------------------------------------------------------
#include <csl_timer.h>

class ExecMeasure
{
private:
    TIMER_Handle hTimer;
public:
    ExecMeasure()  { hTimer = TIMER_open(TIMER_DEV1, 0); }
    ~ExecMeasure() { TIMER_close(hTimer); }
    void Execute1();
    void Execute2();
};

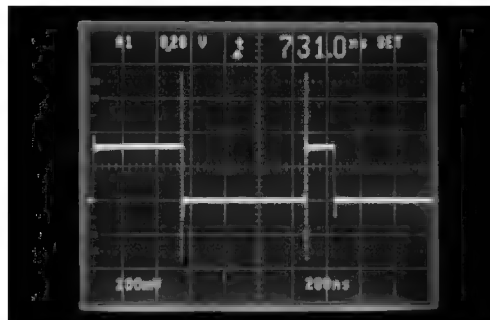
// 幅の広いパルスを発生するためのメンバ関数
void ExecMeasure::Execute1()
{
    TIMER_setDatOut(hTimer, 1);
    TIMER_setDatOut(hTimer, 1);
    TIMER_setDatOut(hTimer, 1);
    TIMER_setDatOut(hTimer, 0);
}

// 幅の狭いパルスを発生するためのメンバ関数
void ExecMeasure::Execute2()
{
    TIMER_setDatOut(hTimer, 1);
    TIMER_setDatOut(hTimer, 0);
}

// パルス発生器に対応するオブジェクトの宣言
ExecMeasure TOUT;
```



写真A.1　実行時間の測定時に，設定箇所にオシロスコープのプローブを当てているようす



写真A.2　オシロスコープで実行時間を測定しているようす(リスト4のプログラムの場合)

表A　IIRフィルタの実行時間

クラスの定義が記述されているリスト番号	フィルタの構成	係数および演算の際に用いたデータの型	実行時間(μs)
1	直接形II	double型	0.72
1	直接形II	float型	0.60
3	縦続形	float型	0.73
5	縦続形	float型	1.38

表2　リスト 4の変更箇所

記述されている箇所	リスト 4	変更後
インクルード・ファイル	IIR_Cascade2nd.cpp	IIR_Cascade1st2nd.cpp
係数のグローバル宣言	IIR_Cascade2nd<ORDER>	IIR_Cascade1st2nd<ORDER>
lpf オブジェクトのグローバル宣言	IIR_Cascade2nd<ORDER>	IIR_Cascade1st2nd<ORDER>

るためのメンバ関数 Execute() が定義されています．

直接形 II のクラスでは，フィルタによって float 型の精度で十分な場合と double 型の精度が必要になる場合がありました．そこで，両方の型に対応できるようにするためにテンプレート・クラスとして実現しました．しかし，縦続形で使う基本単位である 1次または2次の IIR フィルタでは，float 型の精度で十分な場合がほとんどのため，テンプレート・クラスにしていません．それぞれのメンバ関数 Execute() は float 型の精度で実現しています．

▶ 縦続形 IIR フィルタのクラス： **IIR_Cascade1st2nd**

クラス IIR_Cascade1st2nd は縦続形 IIR フィルタのクラスです．このクラスのメンバ関数 Execute() を実行すると，フィルタが実行されます．

縦続形の IIR フィルタは，次数が奇数と偶数の場合でフィルタの基本単位の構成が変わります．そこで，クラス BaseIIR1st2nd のオブジェクトに対するポインタの配列 Hm を宣言し[注8]，コンストラクタの処理でこのポインタに biLinear または biQuad のオブジェクトを割り当てます．

コンストラクタの中では，new 演算子により biLinear のオブジェクトまたは biQuad のオブジェクトを生成し，それをポインタの配列 Hm へ代入します．このようなことが可能なのは，Hm が基底クラスである BaseIIR1st2nd オブジェクトのポインタであり，さらにクラス biLinear と biQuad が基底クラス BaseIIR1st2nd を継承する派生クラスになっているからです．

このクラスでは，フィルタの次数が偶数次の場合，このポインタ Hm にはすべて biQuad のオブジェクトが割り当てられます．一方，フィルタの次数が奇数次の場合，先頭のポインタには biLinear のオブジェクトが割り当てられ，残りのポインタには biQuad のオブジェクトが割り当てられます．

フィルタを実行するためのメンバ関数 Execute() は，ポリモーフィズムを利用しているので，ただ単に for 文で 1段目の出力を 2段目に入力し，2段目の入力を 3段目へ入力し…という処理を行っているだけです．したがって，プログラムが簡潔に記述できるようになります．

● 縦続形のクラスを使った IIR フィルタの実現

クラス IIR_Cascade1st2nd を使って作成した IIR フィルタのプログラムはほとんどがリスト 4と同じなので，変更したところのみを表2に示します．

実行した結果は p.191 の写真 1の上の波形と同じになるので，省略します．

*　　　　*

この連載が C++ を使って DSP のプログラミングを行うことを考えている読者に，何らかのヒントになれば幸いです．

最後になりますが，この連載を進めるにあたって，日本テキサス・インスツルメンツの角田勇氏をはじめ関係各位には，DSK の提供や，DSK を使ううえでのいろいろな相談に応じていただき，たいへんお世話になりましたことを感謝いたします．

注8：クラス BaseIIR1st2nd は抽象クラスなので，

```
BaseIIR1st2nd  Hm[(order+1)/2];
```

という宣言はコンパイル・エラーとなる．

みかみ・なおき　職業能力開発総合大学校 情報工学科

シニアエンジニアの技術草子 四拾参之段

◆医は仁術

旭 征佑

● 何かがまちがっている日本の医療

みぞおちの下あたりにシコリができ，腫れて痛くなってきた．まさか悪性の腫瘍？　勝手にそう思えてきて体調までもよろしくない．周りにもすすめられて，しぶしぶ病院に行く羽目になってしまった．気が乗らないのだが，この際しょうがない．

総合病院に行ったほうがいいだろう．漠然とそう考えて，近くの大学病院に行った．総合受付に行くと，担当者が筆者のシコリを指で触って，さかんに頭をひねっていたが，内線で医師に問い合わせたらしく，内科にまわされることになった．10時ちょうどに内科で受付を済ませ，年甲斐もなく少しドキドキしながら診察を待った．それから待つこと数時間．周りにびっしり座っていた患者たちも次々と姿を消し，筆者だけが取り残されてしらけた雰囲気になってきたころ，やっとのことで診察してもらえることになった．担当の医者は，遅くなったことを筆者にさかんに謝まってはいたが，肝心の診察のほうは歯切れが悪く「内科ではよくわからない」という．結局，心臓外科に行くことを勧められた．ちょっと不満だったが，この日は愚痴はこぼさないことにした．ぐったり疲れて病院を出たときは，すでに2時を回っていた．

朝早く受付を済ませばもっと早く終わるかもしれない．翌日はそう考えて開院の9時ちょっとすぎには心臓外科の受付を済ませた．ある程度の待ちを覚悟して小脇に本を抱えていった．しかし，この考えは大いに甘かった．というのは，その日の診察が終わったのもやっぱり2時過ぎだったからだ．異常な疲れが残ったが，こんなに診察が遅くなるとは，きっと9時前から常連を含む大勢の患者が並んでいたに違いない．

この日はそれだけではなかった．何時間もじっと座っていた間にクーラの冷気で体がすっかり冷え切っていたのだ．どうやら，空いていると思って座っていた席は，クーラの冷気が直接ふきつける場所だったようだ．途中で場所を何度か変わったが，多くの人が診察を終わって帰っていくと，人が減ってクーラがますます強烈に効いてくる．「寒い」とナースにうったえたのだが，全館自動空調なので部分的な温度調節はできないという．

四六時中病人の相手をさせられているナースや医者もたいへんだろう．うつろな目の患者に毎日何時間も見つめられ続けている病院の職員も気の毒といえば気の毒ではある．しかし，2日間，劣悪な環境で，しかもいつ呼び出されるかわからないというストレスの強い状態で都合9時間待たされ続けた．そんな患者のほうもつらいことこのうえない．筆者はまだ若いからいいが，比較的重い病気の人やお年寄りは，筆舌に尽くし難い苦痛を味わっているに違いない．これでは治る病気も治らないではないか．

● 医療の顧客満足度

最近の高度医療の発達もあり，どの病院にも先進的な医療機器が導入されている．地元のちょっとした病院になると，CTスキャンやら，超音波，MRI（磁気共鳴診断装置）など，1台で数億円もする機器がおいてあったりして驚く．診察する医師の机の上にパソコンが乗っていることはそんなに珍しくなくなった．ミミズがはったと揶揄された悪名高き医師の処方箋も，いまやマウスをクリックするだけで正確かつ明瞭にプリンタで打ち出せる時代に変わった．しかも，ディスプレイでは，なんと新薬のHELPまでもあるのだ．大きな大学病院では大掛かりなカルテ管理システムも稼動している．医師を支援するシステムの発達と充実度には目を見張るものがある．しかし，ディスプレイに向かってマウスをいじりながら診察を終えてしまう医師も困りものだ．患者軽視もはなはだしい．だいたい，自分が「医師」で，お客を「患者」と者呼ばわりしている．こんなのは，いったい他のどの業界にあるだろうか．そういえば，顧客である患者の便宜を図るシステムといえば，いったいどんなものが開発されたのだろうか．筆者の知っている限りではあまりない．

「医は仁術」ということばは，教科書にも登場した貝原益軒の「養生訓」に登場する．ここでは「医は仁術なり．仁愛の心を本(もと)とし，人を救ふをもって，志とすべし．わが身の利養を専(もっぱら)に志すべからず」と医師の心構えを説いている．ようするに，医は優しさで，自分のことよりまず患者のことを考えろといっているのだ．貝原益軒が生きていれば，今の病院のシステムは，仁術ではないというに違いない．

● せめて受付ナンバでも発行してくれれば

銀行や郵便局のように，受付ナンバを発行してくれるしくみは，シンプルでいい．あと何人待っているのか一目瞭然だし，だいたいの待ち時間も想像がつくからだ．

身近に小さな医院があるが，数年前に受付ナンバを発行するシステムに変えたようだ．患者は朝早く受付に並び，予約ナンバをもらう．待合室には，現在診察中のナンバを確認できる電光掲示



板がある．これを見て，だいたい何時ごろになるか患者自身が予想し，少し前には待合室に来るようにしているようだ．また，電話で問い合わせると，受付が診察中のナンバを口頭でこたえてくれる無料サービスもあり，たいへん便利だと評判が良い．

大学病院の薬局などでは，薬のできた順に電光掲示板に番号が出るようなしくみが十年以上前からあると認識している．これは，薬ができているかどうか一目でわかるし，あとどれくらい待てばいいかも大体見当がつく．呼ばれるのを待っていて，トイレに行くのも躊躇する，などという必要もない．

このようなシステムは，大きな病院の診察待合室に導入するとなると，勝手が違うかもしれない．同じ科の中でも担当の医師が違ったり，医師の判断で順番を変えることがあるため，必ずしも受付の順番どおりにならないからだ．そうすると，単純に番号制のシステムは運用できないことになる．

● 診察待ち改善システム

科ごとに，診察の進行状況と現在の待ち人数から，予想される診察時刻を試算するシステムはできそうだ．たいていの大病院は，診察券は磁気カードになっている．患者自身が端末のそばにでも行って，診察券をカード・リーダに通せば，「現在の診察の予想時間はXX時XX分ごろです」といってくれるというのはどうだろう．いや，これではカード・リーダを通せないお年寄りもいるだろうし，だいいち25年前の仕様だ．こんなシステムを開発したら，笑われてしまうだろう．

そうであれば，診察券に最近注目の非接触式ICカードを採用すればいい．使用する電磁波もきわめて弱いので，病院内でも十分問題なく使用できるはずだ．カードを持って近づくと，あとどのくらい待つかの目処が出る．これなら，気兼ねなくトイレにいったり，軽く外で一服できる．カルテにも貼り付けておけば，ナースが患者を呼び出すときや，現在の診察の進行状況を自動的にサンプリングすることもできる．

Webサーバと組み合わせれば，近くの喫茶店で携帯に診察券ナンバを入れると待ち時間が試算できる，などということも技術的には決して難しくはないはずだ．

しかし，「医は仁術」を理解できない医師・病院の経営者側の意識改革が最大の障害なのかしれない．ぜひ顧客である患者の満足度を向上させることを考えてみてほしいのだが，そういう



改革をしてくれそうな病院はなかなか見当たりそうにない．

*　　　　　*

余談だが，診察の結果はというと，筆者はみぞおちのあたりに内側から胸骨が突き出ている骨格だそうで，その突き出た骨の上に，大量の脂肪が乗り，あたかもシコリのように見えただけだった．痛くなったのは，触りすぎたせいだという．心臓外科に行くことになったのは，実際に胸を開いた経験がある医者に判断してもらおうということだったらしい．

なんでもないとわかると恥ずかしい話なのだが，そんなことよりも高い治療費を払ったあげく，9時間も劣悪な環境と精神状態で待たされた腹ただしさが後味悪く残った2日間だった．病院に行くと多かれ少なかれいつもこうだ．そうでなかったのは，十年ほど前，港区A病院の一室で，大ファンだった女優の田中美佐子と向かい合って採血待ちをしていた，ほんのわずかなひとときだけにすぎない．このときばかりは，病院に来てよかった思ったものだ．

あさひ・しょうすけ　テクニカル・ライタ
イラスト　森 祐子

電脳事情にし ひがし

猪飼 國夫

国内外に見る研究学園都市とハイテク産業の集中化…中国編（上）

情報産業などハイテク企業が国家の将来を決めるとされ，世界各地で有能な人材を求めて，研究学園都市のような街が形作られています．これは，集中することで人材や情報を集めやすいというハイテク関連特有の事情が働いています．今回はSilicon Vallayを中心とした報告ではなく，Asiaに眼を向けてみたいと思います．

● 産業構造を変えて日本を再生する

日本の経済はこのところ好調にみえますが，それは自動車や素材・部品関係の輸出が好調なことに由来しています．国内の需要はあい変わらずしぼんだままで，そのことが400万人ともいわれるフリータを生み出しています．

新規の卒業者も毎年10万人規模の割合で安定した職を得られないまま社会へ送り出されています．理工系の学卒者ですらあまり芳しくない就職状況が続いています．

新しく雇用を吸収する産業が育っていないとか，中国で安価にものを作るのが原因だとか，日本の若年人口の減少が需要縮小の引き金になっている，などとたくさんの原因追求がなされる中，それらの個々の要因と思われる事がらに対する解決策は一向に見つかっていません．多分，そのどれ一つを取ってみても，成熟して豊かさに慣れてしまった日本社会では，現在の生活を放棄しなければ解決の糸口はつかめないでしょう．

唯一，可能性がある解決策は，雇用を吸収できるような新しい産業を創り出すことです．堺屋太一氏がいうところの知価革命による，20世紀型の大量生産から個々の産物の付加価値を重視した産業への転換などがそれに当たります．

表1　研究学園都市として名乗りを上げているところ（LaboLink2003より）

旭川リサーチパーク，恵庭リサーチ・ビジネスパーク，
八戸ハイテクパーク，盛岡西リサーチパーク，
アルカディアソフトパーク山形，泉サイエンスパーク，
郡山ウエストソフトパーク，茨城県那珂郡東海村，
筑波研究学園都市，ソフトリサーチパーク情報の森とちぎ，
太田リサーチパーク，かずさアカデミアパーク，
横須賀リサーチパーク，湘南国際村，
長岡オフィス・アルカディア，佐久リサーチパーク，
富山イノベーションパーク，いしかわサイエンスパーク，
ソフトパーク福井，上田リサーチパーク，
東濃研究学園都市，テクノポリス都田開発地区，
交流未来都市，志段味ヒューマンサイエンスパーク，
桑名ビジネスリサーチパーク，鈴鹿山麓リサーチパーク，
関西文化学術研究都市，けいはんな，
リヴェール和泉，神戸リサーチパーク（鹿の子台），
播磨科学公園都市，海南インテリジェントパーク，
岡山リサーチパーク，広島中央サイエンスパーク，
鳥取新都市（テクノリサーチパーク），ソフトビジネスパーク，
宇部新都市（テクノセンターゾーン），香川インテリジェントパーク，
ブレインズパーク徳島，北九州テクノパーク，
飯塚リサーチパーク，鳥栖北部丘陵新都市，
オフィスパーク大村，熊本テクノ・リサーチパーク，
大分インテリジェントタウン，宮崎テクノリサーチパーク，
国分上野原テクノパーク，トロピカルテクノパーク

● 研究学園都市

従来型の産業なら，低賃金で働く労働者の確保が立地の大きな条件になっていたのですが，ハイテク産業に転換するためには，高熟練な職人や高学歴な従業員を確保できることが立地の大きな条件となります．そこで，大都市周辺か理工系の大学・研究機関が存在する周辺が候補地になります．

そのような目論見の一つとして，日本では高度成長期の国土開発の波に乗り，米国Calfornia州のStanford大学を中心としたハイテク産業の集積地Silicon Valleyを模して，現つくば市に研究学園都市が作られました．東京教育大学を改組して筑波大学として移転するとともに，おもに工業技術院傘下の国立試験研究機関を核として，ハイテク産業を集めようとしたのです．

ここでの産学協同による新産業の創設の成果は，研究学園都市の完成後20年以上経った今でも定かではないにもかかわらず，国内では同じような考えから，いくつかの同じような構想が形を変えて提案・実施されました．実現したものや計画中のものを含めると十指に上ります．

京阪奈（京都・大阪・奈良）にまたがる関西文化学術研究都市や北九州の旧筑豊炭田地帯の再生をめざした飯塚地区などが大規模な情報産業の集積地としてできあがりつつあります．また，細かいものでは表1に示すように，いろいろな地域が旧来の製造業中心の産業振興では中国などの外国の安価な労働力と対抗できないため，情報産業やナノテク，バイオテクなどのハイテク産業に活路を見い出すそうとしています．なぜか「何々パーク」という名称が多いようです．

● 自然発生的な情報産業の集中

一方でこのような行政が主導する研究学園都市ではなく，東京の笹塚や秋葉原など，情報産業や電脳関係の商店が自然と集まった街もあります．どちらかというと，これらの街は自然発生的で計画性がない分,「何とか都市」や「何とかパーク」と名付けることが難しいうえに，街の内容も非常に流動的です．しかし段違いに活気があり，新規の企業が生まれ育ち，変遷しています．

秋葉原などは，電子部品街→家電街→パソコン街→ソフト街→オタク街（?）と短期間で大きく変遷を遂げていて，新旧の街が混在しています．私もバブルで値上がりしたオフィスの賃料を避けて，荒川を渡って避難するまでは，1975年の創業以来，秋葉原とその周りで仕事をしていました．

● 欧米での状況

欧米に眼を向けると，やはり企画した都市と自然発生的な街の2種類があることがわかります．前掲のSilicon ValleyはStanford大学によって企画された街ですが，Calfornia州立大

学の Berkeley 校舎も数十 km 以内にあり，集積と拡大はかなり自然に行われました．

欧米のふるい街には旧来の学園都市が散在します．米国東部の街 Boston の MIT，Harvard 大学や英国の Cambridge 大学周辺は学園都市となっていますが，情報産業が集中しているわけではありません．仏 Paris 郊外の Ile de France は大学や研究機関が集まっている学園都市です．南仏 Nice の Sophia Antipolis は情報関係の学校や企業が多く，ハイテク基地といえるでしょう．

● Asia の状況

情報産業が集積しているところとしては，約 1,200 社があるといわれる南印 Bangalore が有名です．日本企業も多く，日本人会があるくらいです．

人件費の上昇で，日本に代わる製造業の基地としての地位が危うくなった韓国では，日本の地方振興策と同じような意図をもって「産業技術団地(テクノパーク)支援に関する特例法」を作成し，1999 年から大邱テクノパークや慶北テクノパークなど 6 か所を策定しています．

北京の北西には有名な中関村があり，電脳街から発展して非常に多くの情報産業が集まっています．この街は，中国一の理工系大学といわれる精華大学をはじめとして，北京大学，中国人民大学など多くの有名大学を抱える環状四号道路の付近に自然発生的にできました．台北でも台北科学技術大学の周辺に自然発生的な電脳街が拡がっています．

日本企業と中国の関係からいうと中国最大のソフトウェア企業東軟集団を産み出した理工系の東北大学(瀋陽市)の影響下にある大連には，多くの日本企業が進出しています．データ打ち込みや画面の作成のような日本語の処理を行う人材も豊富で，日本語教育も盛んです．市や大学の Web サイトも日本語で書かれているページもあるくらいです．

● 情報産業を中心とした雇用の創出

このように，日本や世界の状況を見渡すと，きっかけはどうあれ，情報産業などが集中するハイテク都市にはいくつかの特徴があると思われます．すなわち，

1) 気候が過ごしやすい
2) 都会的な雰囲気がある
3) ある程度の数の大学がある
4) 交通が便利である

などの点です．

気候が良いということは，高度な技術を持つ技術者にとって住みやすいだけでなく，そこで使われる機器や生産される機械にとっても条件が良いということです．

高度な知識とシステム構築力を持った人材は，当然ながら高い給料を必要としますが，それ以上に快適な生活空間を求める傾向にあります．余暇を過ごすための施設だけではなく，日常的に同水準の人達との交流が図れる必要もあります．そのためには，ある程度の人口も必要となります．Bangalore は人口 5,300 万人の Karnataka 州の州都で，Deccan 高原にあるため，比較的良好な気候に恵まれています．



写真1　案内してくれた三通の周師偉社長(左)とスタッフ

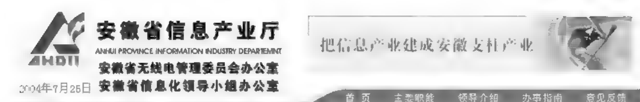


図1　安徽省情報産業庁ページ(http://www.ahdii.gov.cn/)

● ケース・スタディ

ハイテクの街を作って産業を発展させようという試みは，このように世界中で試みられていて，成功した事例もあれば，意気込みだけが先行した事例もあるようです．

われわれは，そのような街のどこへ行けば快適に仕事ができるでしょうか．海外の話としては，Silicon Valley の話だけがたくさん伝わっているようです．次回は，中国の合肥(ガッピ[注1])での世界有数のハイテク基地を作ろうという試みを，現地を訪問し，入手した資料から将来を占ってみたいと思います(写真 1)．

なお，現地側の詳しい情報は，大連のように日本語では読めませんが，図 1 をご覧ください．

いかい・くにお　(株)エム・アイ・ベンチャー

注1：中国側は合肥を「ゴウヒ」と読ませたがっているが，それだと合併や合算も「ゴウヘイ」，「ゴウサン」と読まなければならなくなり，日本語の発音体系に反してしまう．

NEWPRODUCTS

●64ビットRISC型マイクロプロセッサ

TX9956CXBG-666 TX9956CXBG-533

- 90nmのプロセスを採用し，ミップスのRISCアーキテクチャを利用したオリジナル・プロセッサ・コア「TX99/H4」を搭載.
- 「TX9956CXBG-666」は最高動作周波数666MHz，「TX9956CXBG-533」は533MHzの高速処理が可能.
- 256Kバイトの大容量セカンダリ・キャッシュ・メモリを内蔵しているため，キャッシュに貯えられているプログラムやデータなどのヒット率が向上.
- ハイエンドのレーザ・プリンタやセット・トップ・ボックスなどのディジタル情報機器に適する.

●サンプル価格：¥5,000

■（株）東芝
TEL：03-3457-3491

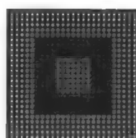
●32ビット・マイクロプロセッサ

SH7780

- 最大動作周波数400MHz，720MIPSの処理性能を実現した「SH-4A」CPUコアを搭載し，32ビットPCIバス・コントローラを内蔵.
- キャッシュ・メモリは，4ウェイ・セット・アソシアティブを採用し，命令用とデータ用をそれぞれ32Kバイト内蔵.
- 最大400MHzで動作する，浮動小数点演算器（FPU）を搭載.
- FPUは，単精度および倍精度演算をサポートし，演算処理性能は単精度で最大2.8GFLOPSを実現.
- DDR-SDRAMと接続するための32ビット・バス，PCIバスに接続するための32ビット・バス，フラッシュ・メモリやSRAMなどと接続するための32ビット・ローカルバスの3種類のバス構成を採用.

●サンプル価格：¥6,000





■（株）ルネサス テクノロジ
TEL：03-5201-5219

●MMU搭載プロセッサ・コア

MB93461 MB93441 MB93443

- 「MB93461」は，MMU搭載FR-Vプロセッサ・コア「FR450」とコンパニオン・チップ「MB93495」の持つビデオ，オーディオなどの入出力機能に，USBホストを1チップ化したSoC.
- 同社の従来のSoC製品と比較して，2倍の処理性能と，1/2以下の消費電力を実現.
- QVGAサイズのMPEG-4録画で，30フレーム/sの処理が可能.
- VGAサイズのMPEG-4エンコード/デコードが可能なため，テレビ，高機能プロジェクタ，IPテレビ電話機，携帯メディア・プレーヤなどのディジタルAV機器に適する.
- FR-Vファミリ・プロセッサと直結可能なブリッジ・チップ「MB93441」，「MB93443」を提供し，LANやUSB，マイクロドライブ，PCカードなどの周辺インターフェース機能の拡張を実現.

●サンプル価格：¥4,500（MB93461）
¥1,350（MB93441）
¥1,050（MB93443）

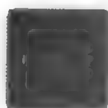
■富士通（株）
TEL：03-5322-3354
E-mail：edevice@fujitsu.com

●16ビット・マイコン

M16C/6NK M16C/6NL

- 16ビットCPUコア「M16C/60」を搭載し，動作電圧は3.0～5.5V.
- プログラム格納用途のフラッシュ・メモリを最大512Kバイト，RAMを最大31Kバイト内蔵することで，プログラム規模の増大や処理速度の高速化に対応.
- フラッシュ・メモリ容量は，512Kバイトと384Kバイトの2種類を用意しており，システム構成に合わせた選択が可能.
- プログラムとデータ格納に使用可能な4Kバイトのフラッシュ・メモリを搭載しているため，外付け部品の削減が可能.
- CAN対応コントローラを「M16C/6NK」は2チャネル，「M16C/6NL」は1チャネル内蔵.
- 多機能タイマやシリアルI/O，I^2Cバス，A-D/D-A変換器，DMAコントローラ，CRC演算回路，監視タイマなどの周辺機器を搭載.

●サンプル価格：¥1,530～¥1,750





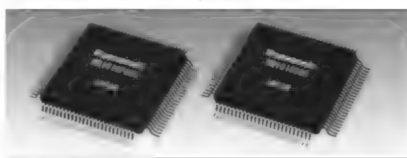
■（株）ルネサス テクノロジ
TEL：03-5201-5235

●8ビット・マイコン

MN101C86G MN101CF86G

- DVDレコーダなどのディジタル録画機器に適した全世界対応VBI（Vertical Blanking Interval）データ・スライサを搭載した8ビット・マイコン.
- 電波状態のよくない環境でも，最適なスライス・レベルを設定し，安定したスライス結果を得る弱電界対策回路を搭載.
- 同期式/UART兼用（5本）およびI^2C（1本）シリアル・インターフェースを内蔵することで，I^2Cでチューナと接続，同期式/UART兼用でDVDバックエンドLSI，VTRサーボLSI，FLドライバIC，シアタ・アンプICなどと接続したシステム構成が可能.
- スライスしたデータを，CPUコアを介さずに直接シリアル・インターフェースを経由して他のLSIに送信するためのDMA機能を搭載.

●サンプル価格：¥2,000



■松下電器産業（株）
TEL：075-951-8151
E-mail：semiconpress@scd.mei.co.jp

●8ビット・マイコン

S1C8E108

- 温度湿度計測電波タグ向け8ビット・マイコン.
- 抵抗周波数変換型A-Dコンバータ（2チャネル）を内蔵しており，抵抗とサーミスタ，あるいは湿度センサを外付けするだけで，温度，湿度の計測が可能.
- 抵抗周波数変換型A-Dコンバータの採用により，一般のA-Dコンバータと比較して，1/100～1/1000以下の低消費電力での計測が可能.
- 1024ビットのEEPROMを搭載することで，コンフィグレーション設定，ID書き込み，および温度湿度情報などの高精度な履歴管理システムを実現可能.
- 符号なし整数乗除算回路を内蔵しているため，複雑な演算処理を低負荷で実現.
- 消費電力は，動作時（4MHz/3V）で1.6mA（typ.）.
- 電源電圧検出回路（SVD）を内蔵.

●サンプル価格：¥500

■セイコーエプソン（株）
TEL：042-587-5816
URL：http://www.epsondevice.com/

●16ビット・ディジタル・シグナル・コントローラ

dsPIC30Fシリーズ

- 20～30MIPSの処理速度を備え，フラッシュ・メモリによる自己プログラミング機能を持ち，工業用温度範囲および拡張温度範囲に適応．
- 16ビット・フラッシュ・マイコンの性能と，ディジタル・シグナル・プロセッサの計算能力およびスループット能力を備える．
- エンハンスト・フラッシュに搭載されている自己プログラミング機能により，遠隔地からフラッシュ・プログラム・メモリをアップグレードして，エンド・ユーザのアプリケーション・コードを書き換えることが可能．
- 「dsPIC30F2010」および「dsPIC30F6010」は，モータ制御用のパルス幅変調モジュールと500Kspsの10ビットA-Dコンバータを搭載．
- 「dsPIC30F6011」,「dsPIC30F6012」,「dsPIC30F6013」,「dsPIC30F6014」は，132～144Kバイトのエンハンスト・フラッシュ・メモリと6～8KバイトのSRAMを搭載．

●価格：下記へ問い合わせ

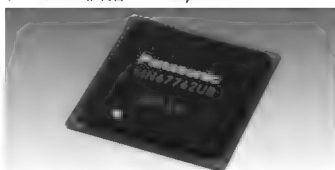
■マイクロチップ・テクノロジー・ジャパン（株）
URL：http://www.microchip.com/dspic/

●グラフィック・プロセッサ

MN67762

- グラフィック機能として，グロー・シェーディング，αブレンド，フォグ，スペキュラ，スポットライト表現などをサポート．
- 光沢や明暗を持たせるなどの自由度の高い，多彩で美麗なグラフィック効果を，1677万色で高速に表現することが可能．
- 図形境界を滑らかにするアンチエイリアスを，ハードウェアで高速に処理することが可能．
- 高速にデータ転送が可能なDDR-SDRAMに対応（内部バス幅：64ビット）することにより，従来品と比較して最大2倍の描画性能向上を実現．
- 回路で光源処理演算を行うことで，図形の輝度を高速に生成し，CPUの処理負荷を低減．

●サンプル価格：¥10,000



■松下電器産業（株）
TEL：075-951-8151
E-mail：semiconpress@scd.mei.co.jp

●CPLD

MAX IIデバイス・ファミリ

- ルックアップ・テーブル・ベースのCPLDアーキテクチャを基に開発され，従来のMAXファミリと比較して，半分のコストと1/10の消費電力を提供．
- シングル・チップ，不揮発性，優れた操作性などの特徴を維持．
- 従来製品の4倍の集積度と2倍以上の性能を実現．
- 240個から2210個のロジック・エレメント集積度を網羅する四つの製品で構成．
- デザイン・ソフトウェアであるQuartus IIによりサポートされる．
- 32ビット・エッジ・コネクタ，プログラミング/USBケーブル，複数のデモンストレーション・デザインが付属するPCI形状プリント回路基板などを含む，EPM1270を搭載した開発キットを販売予定．

●価格：$1.50～$7.00（500,000個時）

■日本アルテラ（株）
TEL：03-3340-9415 FAX：03-3340-9487

●USB2.0プログラマブル・コントローラ

EZ-USB FX2LP（CY7C6803A-X）

- 480Mbpsのデータ・レート，16Kバイトのオンチップ・メモリ，最大40のプログラマブルI/Oを特徴としており，設計においても柔軟性を提供．
- パッケージ・オプション中，省スペース8×8mm QFNパッケージは，フォーム・ファクタの小さいモバイル・アプリケーションに適する．
- ピン配列は,「EZ-USB FX2」と互換．
- サポートしているファームウェア，ドライバおよびリファレンス設計ツールは，汎用USB製品やATA特定製品をターゲットとするだけでなく，アイソクロナス・データ転送，ビデオ・ストリーミング，Media Transfer Protocol（マイクロソフト）を用いる製品など多岐にわたる．
- 独自開発の低消費電力USBを用いることで，他社製品と比較して，消費電力を約50%低減．

●サンプル価格：$4.45～（10,000個時）

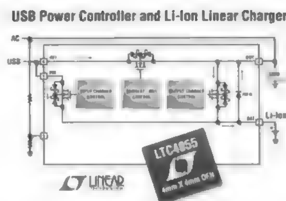
■日本サイプレス（株）
TEL：03-5371-1921 FAX：03-5371-1955

●パワー・マネージャ/バッテリ・チャージャ

LTC4055

- USB周辺機器に電力を供給し，USB V_{BUS}やACアダプタ電源から，周辺機器の1セル・リチウム・イオン・バッテリを充電．
- USB電流制限仕様に準拠しており，システム負荷電流が増大するにつれ，バッテリ充電電流を自動的に低減．
- バスが接続されている間，完全に充電されたバッテリをフレッシュに保つために，バッテリから電力を取らず，USBから負荷に電力を供給．
- 電源が取り外されると，内蔵の低損失ダイオードを介して，バッテリから負荷に電流が流れるため，電圧降下と消費電力を最小限に抑えることが可能．

●サンプル価格：¥254（1,000個時）



■リニアテクノロジー（株）
TEL：03-5226-7291 FAX：03-5226-0268

●バッテリ・マネジメントIC

bq24100

- 小型パッケージに2Aの充電を供給するパワーFETを内蔵し，省スペースでより大きな充電電流を実現．
- 発熱量を大幅に削減することで，システム設計を容易にし，ポータブル・システムに内蔵される1～2セル構成のLi-IonまたはLi-Polyバッテリの充電回路に適する．
- 1.1MHzの固定スイッチング周波数により，小型インダクタの使用を可能とし，低リプルで16Vまでの入力電圧で動作．
- 高精度CVCCバッテリ充電を行い，充電過程をコントロールする複数の充電ステータス出力，プリチャージ，電池検出，低電圧再充電機能，および充電終了動作を提供．
- スタンドアロン動作およびシステム制御動作の各バージョンを供給．
- LEDまたはホスト・プロセッサ・インターフェース用の複数のステータス出力を用意．
- 絶対最大入力耐圧は20V．

●価格：$2.10（1,000個時）

■日本テキサス・インスツルメンツ（株）
FAX：0120-81-0036
URL：http://www.tij.co.jp/pic/

■弊誌では新製品に関するニュース・リリースを募集しております．
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュース・リリース係
FAX：（03）5395-2127，E-mail：mngnews@cqpub.co.jp （編集部）

●UHF RFID チップ

XRA00

- RFIDリーダから発信され，付属の小型アンテナで受信される電磁波のエネルギで動作.
- 近傍型デバイスに分類され，リーダから最大10m離れた地点からの読み取りが可能.
- UHF技術は，902～928MHz(米国)，および866～868MHz(欧州)の2通りの帯域に対応.
- 不揮発性メモリの搭載により，サプライ・チェーンでのタグ貼り付け時にプログラミングして，個別アイテムごとに所定のコードの入力やデータの更新が可能.
- 128ビット・メモリは8ブロックに分割され，先頭のブロックにはEPC標準に基づく16ビットの巡回冗長検査(CRC)データ，中央の6ブロックには実際に使用される96ビットの製品コードが格納される.

●サンプル価格：¥6(5,000,000個時)



■STマイクロエレクトロニクス(株)
TEL：03-5783-8240 FAX：03-5783-8216

●チャージ・ポンプIC

TPS60230

- インダクタが不要で，白色LED搭載のカラー・ディスプレイ・バックライト・アプリケーションの電源に最適.
- 3mm角のパッケージで87%の変換効率を提供し，最高5個の白色LEDを最大25mA/個の定電流で駆動し，総合出力電流125mAを出力可能.
- 1MHzのスイッチング周波数により，容量が1μF以下の小型コンデンサの使用が可能.
- 各LEDの輝度レベルは2個のイネーブル入力を使用し，3段階の電流調光あるいはPWM調光によって調整が可能.
- 動作入力電圧範囲は，2.7～6.5V.
- ソフト・スタート機能を内蔵し，突入電流を制限.
- 入力リプルおよびEMI(電磁輻射)が低い.

●サンプル価格：$1.55(1,000個時)

■日本テキサス・インスツルメンツ(株)
FAX：0120-81-0036
URL：http://www.tij.co.jp/pic/

●ショットキー・バリア・ダイオード

PicoSBDシリーズ

- 独自のECSP 1608-4外形(1.6×0.8×0.6mm)に搭載し，定格電流I_O=1Aを実現.
- 超低抵抗材料を用い，不純物プロファイルの最適化を図り，従来品と比較してVFを約30%低減.
- 低VFタイプ，低VIタイプの2種類の特性タイプを用意し，用途に応じた選択が可能.
- 環境面に配慮し，完全鉛フリーに対応.
- 裏面端子2ピンを四つに分割した2ピン-4パッドで，実装時の傾きや浮きを防止.

●サンプル価格：¥40



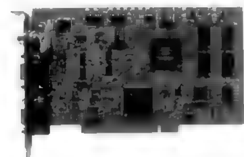
■三洋電機(株)
TEL：0276-61-8055 FAX：0276-61-8854

●リアルタイム画像入力ボード

FDM-PCI 4TS

- TVフォーマット，ノンTVフォーマットのモノクロ・アナログCCDカメラからの画像取り込みに対応.
- ドライバ，キャプチャ・ソフト，SDKを標準で添付.
- 新EIAJ準拠の「カメラ用12ピン・コネクタ」から，CCDカメラへの電源供給が可能.
- 電源ボックスが不要となり，配線の引き回しが簡素化される.
- トリガの入出力機能を装備しているので，各種センサなどの計測機器から出力される信号に合わせて，目的とする瞬間の画像を確実に取り込める.
- 1台のPCで複数枚のボードを同時に搭載することが可能.

●価格：¥134,400



■(株)フォトロン
TEL：03-3238-2106 FAX：03-3238-2109
E-mail：image@photron.co.jp

●Linuxボード

Armadillo-9

- Linuxに対応した，「ARM9」CPUボード.
- プロセッサとして「EP9315」(シーラスロジック社)を搭載することで，処理能力を従来品の3倍以上に高速化.
- ビデオ出力機能とUSBインターフェースを搭載.
- ビデオ出力機能により，画面付きのシステム端末などのシステム構築が可能.
- USBインターフェースへの対応により，市場に流通するUSB機器を使用した拡張が可能.
- メモリの倍増，ハードディスク・インターフェースの搭載，浮動小数点演算機能を強化.
- システム・クロックは，CPUコア・クロックが200MHz，バス・クロックが100MHz.
- メモリはSDRAM 64Mバイト，フラッシュ・メモリ8Mバイトを搭載.
- 10Base-T/100Base-TのEthernetをサポート.
- RS-232-Cシリアル・ポートを2チャネル搭載.

●価格：¥47,250(量産向けモデル)
¥50,000(開発者向けモデル)

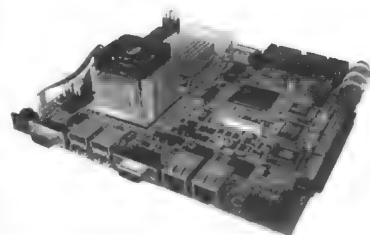
■(株)アットマークテクノ
TEL：011-890-6551 FAX：011-890-6552
E-mail：info@atmark-techno.com

●組み込み用メイン・ボード

HPU7100eMeシリーズ

- CPUにPentium Mを搭載し，200×200mmの小型サイズにPCの機能を凝縮.
- 電解コンデンサを基板上からなくし，寿命部品の経年変化による故障を防止.
- 信頼性の高いECC付きメモリの使用が可能.
- 各種の専用RAL機能を搭載しているため，異常発生時にもその影響を最小限に抑えられる.
- 大型シリンダ電池を採用し，部品交換の手間を軽減.
- コネクタ変更やBIOS変更などの，各種カスタマイズにも対応.

●価格：下記へ問い合わせ



■萩原電気(株)
TEL：052-936-4051

●ボード・ソリューション

NS9750 ボード・ソリューション

- アットマークテクノ社と共同開発した，32 ビット・ネットワーク・プロセッサの Linux 搭載ボード・ソリューション．
- ARM9コアとして性能の高い「ARM926EJ-S」を採用．
- 200MHz の処理能力を持ち，8K/4K バイトの命令/データ・キャッシュを搭載し，MMU をサポート．
- DSP 命令拡張，Java アクセラレータに加え，PCI，Ethernet，USB（ホスト/デバイス），LCDコントローラ，シリアル・インターフェースをワンチップ化．
- 開発キットには Linux のソース・コードを含めた開発環境を用意．
- 高速転送可能な 100Base-TX の LANインターフェースを搭載．
- CardBus スロットを標準搭載することで，広いバス帯域を必要とする IEEE802.11a/11g の無線 LAN 対応が可能．
- ストレージとして使用できる，コンパクト・フラッシュのスロットを装備．

●価格： ¥198,000（開発キット）

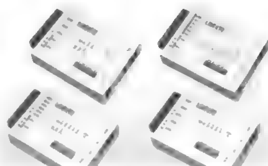
■ネットシリコン ジャパン（株）
TEL：03-5428-0261 FAX：03-5428-0262

●ディジタル I/O ユニット

LANIO シリーズ

- PC から 10/100Base-TX の LAN ネットワーク経由で，遠隔地のディジタル入出力信号を監視，制御可能．
- 必要な入出力点数に応じて，4モデルから選択可能．
- 試運転作業やメンテナンスに配慮した入出力表示 LED と着脱式端子台を装備．
- LANインターフェース部に XPort（ラントロニクス社製）を内蔵．
- 10/100Base-TX に対応しており，TCP/IP，UDP/IP，ARP，HTTP などのネットワーク・プロトコルをサポート．
- PC から簡単な制御コマンドを送ることで，入出力の制御が可能．
- 入力信号の変化で，事前登録したメール・アドレスに E メールを自動送信可能．

●価格： ¥39,900



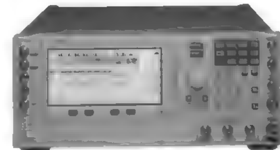
■（株）ラインアイ
TEL：075-693-0161 FAX：075-693-0163

●信号発生器

PSG-D シリーズ マイクロ波・ミリ波 信号発生器

- ベクトル変調信号発生を，44GHz まで拡大することに成功した信号発生器．
- K/Ka 帯を使用した通信衛星や FWA などのベクトル変調信号を，アップコンバータなどの機器を使用することなく，出力を可能にする．
- 最大周波数レンジを，最大 67GHz に拡大．
- サポートするすべての周波数レンジにおいて，高水準の信号出力パワーと位相雑音特性を実現．
- ベクトル変調モデルに，「広帯域変調オプション」を追加することで，1GHz までの超広帯域 I/Q 変調を可能にする．
- 同社のミリ波モジュールとの併用により，70～110GHz の信号出力が可能．

●価格： ¥2,200,000～（CW/アナログ変調モデル）
¥7,500,000～（ベクトル変調モデル）



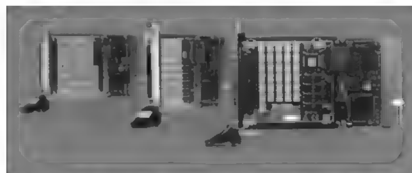
■アジレント・テクノロジー（株）
TEL：0120-421-345

●スイッチ・モジュール

NI PXI-2532

- 最大 512クロスポイントのマトリクス・スイッチングが可能な，高速マトリクス・スイッチ・モジュール．
- 1台の PXI シャーシに最大 4×2176（1ワイヤ）マトリクスを構築することが可能．
- スイッチ管理ソフトウェアである「NI Switch Executive」と組み合わせて使用することにより，チャネル数の少ない高周波数プロトタイプ特性試験から，チャネル数の多い高スループット評価試験や製造試験に至るまで，幅広いアプリケーションに対応可能．
- 汎用的なリード・リレー・スイッチの 2倍以上の速さの 2,000 サイクル/s の高速スイッチングを達成しており，タクト時間短縮を実現．

●価格： ¥662,000



■日本ナショナルインスツルメンツ（株）
TEL：0120-527196 FAX：03-5472-2977
E-mail：salesjapan@ni.com

●JTAG ダウンローダ

JTAG BLAZER

- 専用のハードウェアで構成された，小型の JTAG ダウンローダ．
- FPGA/CPLD 用コンフィグレーション・データを，最大約 12.5Mbps でダウンロードすることが可能．
- 10Base-T/100Base-TX に対応しているため，パラレル・ケーブルや USB ケーブルによるダウンロードと異なり，ケーブル長による制限を受けず，ネットワークに接続している遠隔 PC からのダウンロードが可能．
- コンフィグレーション・データの保存が可能で，保存されたデータは，PC を使用することなくボタン操作だけでダウンロード可能．
- 信号電圧は 2.5～5V で，幅広いデバイスに利用可能．

●価格： ¥37,800

■（株）アットマークテクノ
TEL：011-890-6551 FAX：011-890-6552
E-mail：info@atmark-techno.com

●オンチップ・デバッギング・エミュレータ

E8 エミュレータ

- R8C/Tiny シリーズ向けオンチップ・デバッキング・エミュレータ．
- 「R8C/14」，「R8C/15」，「R8C/16」，「R8C/17」グループのマイコンとの通信は 1本の信号線だけで行い，ユーザ・システムでは使用しないピンを利用．
- 実際の動作に近い状態でシステム・デバッグを行え，デバッグ用に占有するピン数を削減したことで，デバッグを考慮したシステム設計への制約事項を軽減．
- 外形サイズは 97×65×22mm で，ホスト・コンピュータは Windows 対応 PC が可能であり，バス・パワードの USB で接続．

●価格： ¥12,500



■（株）ルネサス テクノロジ
FAX：03-3270-3277
E-mail：csc@renesas.com

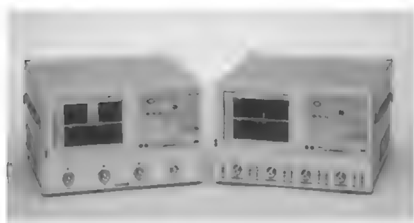
■弊誌では新製品に関するニュース・リリースを募集しております．
宛先は，〒170-8461 東京都豊島区巣鴨 1-14-2 Interface 編集部ニュース・リリース係
FAX：(03)5395-2127，E-mail：mngnews@cqpub.co.jp （編集部）

●ネットワーク・アナライザ

N5230A オプション 240 N5230A オプション 245

- 4μs/ポイントの測定速度を実現.
- 無線機器における実使用時を想定し，内部回路の歪みを把握するために，デバイスへの大パワー印加，デバイスからの大パワー受信を可能にしつつ，フル*N*ポート校正により測定確度を犠牲にしない拡張可能テストセットを，4ポート・ネットワーク・アナライザとして搭載.
- ディジタイザを含むレシーバを改良することで，最高速での測定においても 0.009 dBrms のトレース・ノイズを実現.
- 測定ダイナミック・レンジは，132dB（代表値： 10MHz～4GHz）および 115dB（代表値： 15GHz～20GHz）.

●価格： ¥7,980,000～



■アジレント・テクノロジー（株）
TEL ： 0120-421-345

●無線モデム装置

VDM-1220F

- 特定小電力無線局標準規格に準拠した，1200MHz 帯データ伝送用の組み込み型無線モデム装置.
- チャネル自動選択機能を有しており，ほかの端末が通信中でも自動的に空いているチャネルに切り替えて混信を防止.
- 通信チャネルとして，20チャネルのデータ・チャネル，1チャネルの制御チャネルが使用可能.
- 送信データを一つのパケットとして，1回の送信で相手局に送信.
- 入出力インターフェースは，RS-232-C に準拠し，CMOS シリアルの端末に接続可能.
- 通信速度は，2400/4800/9600/19200/38400 bps のいずれかから選択可能.
- 秘話性として3桁のキー・コードを設定できるため，キー・コードが一致したモデム間のみのデータ伝送が可能.

●価格： オープン

■（株）スタンダード
TEL ： 03-3719-2231

●SIP 専用セキュリティ装置

Applico SIP ファイアウォール

- SIP のヘッダを完全にチェックすることができるため，SIP の NAT 越えを実現し，RTP パケットが使用するポートを理解し，制御することが可能.
- TLS 暗号に対応しているため，SIP の暗号通信を実現.
- SIP URI，MIME セキュリティ機能により，どのアプリケーション利用を許可するかなど，コンテンツ・レベルでのセキュリティを確保.
- 既存のファイアウォールと連携して使用できるため，資産の有効利用が可能.
- DMZ ポートに接続して使用することができるため，ネットワーク構成の変更が不要.
- プロキシ機能により，外部の SIP サーバとの連携が可能.
- SIP レジストラ，SIP Proxy 機能をもつ.

●予定価格： ¥580,000

■（株）アズジェント
TEL ： 03-5643-2561 FAX ： 03-5643-2571
E-mail ： info@asgent.co.jp

●高集積化マルチチャネル CODEC

WM8777-CODEC

- 高機能4チャネルのディジタル・インターフェース・トランシーバ（S/PDIF）を備えた高性能8チャネル DAC と 2チャネル ADC を内蔵.
- 5.1アナログ入力およびディジタル再生ソースの両方に対するバス・マネージメント，ステレオ・ミキサ，トーン＆ボリューム・コントロール機能をチップ内に内蔵.
- 補助オーディオ・インターフェースを備え，CODEC オーディオ・インターフェースから独立して S/PDIF インターフェースの実行が可能.
- 八つの DAC チャネルそれぞれにマスタ・ボリューム・コントロールがあり，ゲイン・レンジは，＋20dB～－100dB まで 1dB ステップで制御可能.
- サンプリング・レートの範囲は 8kHz～192kHz で，内蔵ステレオ・ヘッドホン・アンプにはソース・セレクタを内蔵.

●サンプル価格： $7.34（10,000個）

■ウォルフソン・マイクロエレクトロニクス・ピーエルシー社
TEL ： 03-5328-1400 FAX ： 03-5328-1403

●開発キット

CY4632

- アジレント・テクノロジー社と共同開発の WirelessUSB LS キーボード/マウス・リファレンス開発キット.
- WirelessUSB 光学マウスが付属しており，4ms 未満と短い遅延時間と滑らかなカーソル動作を実現.
- WirelessUSB 干渉排除テクノロジを使用しているため，過密な 2.4GHz 環境でもキーボード入力が確実に感知されるマルチメディア・キーボートが付属.
- 小型のフォーム・ファクタ・ドングルは，WirelessUSB チップと低速 USB enCoRe コントローラを組み合わせたもので，USB プラグ＆プレイ動作が可能で，カスタム・ソフトウェア・ドライバは不要.
- システム・トレイからバッテリ・パワー/信号強度インジケータ，NUM，SCROLL，CAPS LOCK を表示できるため，専用 LED を設ける必要のないユーティリティ・ソフトウェアを付属.

●価格： $199

■日本サイプレス（株）
TEL ： 03-5371-1921 FAX ： 03-5371-1955

●プラットホーム・ソフトウェア

RSA BSAFFE SSL-C

- 暗号化エンジンに FIPS 140-2認定を取得している暗号化モジュールを包含する，SSL 機能組み込みのツール.
- 各種プラットホーム，OS に最適化されており，高水準の API を提供.
- インターネットに接続可能な携帯電話や PDA，カーナビ，キオスク端末，POS，パーキング・ゲートなどの組み込みデバイス，およびルータなどをビジネス活用するためのプラットホームである「NetConscious Ver.1.1」の暗号化通信機能の実装に採用.
- ライセンシを利用することで，各種の技術標準に準拠した製品を短期間で開発でき，知的財産，所有権などの法的安全を確保できる.

●価格： 下記へ問い合わせ

■RSA セキュリティ（株）
TEL ： 03-5222-5210
E-mail ： info-j@rsasecurity.com

●ARM用組み込み向けワークベンチ

IAR Embedded Workbench for ARM 4.11A

- C99機能として，インライン・キーワード，同一のスコープ内での宣言文と命令文の混合，forループの初期化式内での宣言文を可能とした．
- CMX TinyおよびRTOSプラグインを製品に同梱．
- Macraigor USBdemon JTAGインターフェースをサポート．
- ジェネリック・フラッシュ・ローダ機能を改善し，カスタム・フラッシュ・ローダの開発が容易．
- アナログデバイセズ，フィリップスのいくつかのフラッシュ・ローダを追加．
- アナログデバイセズ，アトメル，シーラスロジック，フリースケール・セミコンダクタ，フィリップスのいくつかのI/Oレジスタ定義ファイルを追加．
- 製品のライセンス管理システムは，パラレル・ポート・ドングルに加え，USBドングルを正式にサポート．

●価格：下記へ問い合わせ

■IARシステムズ（株）
URL：http://www.iarsys.co.jp/

●ソフトウェア・ツール

SQMlint

- 自動車用ソフトウェア開発を目的とした，C言語でのプログラム作成時の記述ガイドライン「MISRA Cルール」に基づいて，C言語のソース・コード記述を自動的に検査．
- 「M32R」，「M32C」，「M16C」に加え，「SuperH」および「H8」に対応．
- 気づきにくい記述ミスやプログラム実行時に予期しない動作を引き起こす記述ミスなどの検出が可能．
- Cコンパイラに追加し，作成中のプログラムのコンパイル時にルール・チェックを指定することで，検査を自動的に行い，問題となる記述箇所の結果を出力．
- スピーディな自動検索と手軽に修正や確認が可能であることから，ソフトウェア開発の前工程段階でバグを早期対策できる．

●価格：¥180,000



■（株）ルネサスソリューションズ
TEL：03-5201-5022

●IPv4/IPv6デュアル・スタック

PrCONNECT/Dual

- IPv4/IPv6の二つのホスト・アドレスを持ち，両プロトコルに対応可能な組み込み機器の開発が可能．
- 各種プロトコルを標準で提供．
- IPv6は「ITRON TCP/IP API仕様」を拡張した独自APIを提供．
- IPv4のAPIは，「PrCONNECT2」と互換性を持っているため，アプリケーションの移行をスムースに行える．
- ドライバ依存部は「PrCONNECT2」と互換性を持っているため，「PrCONNECT2」用のドライバをほとんど変更することなく使用可能．
- コンフィグレーション時の機能選択スイッチにより，必要十分なコード・サイズまで縮小可能．
- IPv4動作において，IEEE802.11bドライバ，Webサーバ「PrHTTPD」，メール・クライアント用ライブラリ「PrMAIL」と合わせて使用することが可能．

●価格：下記へ問い合わせ

■イーソル（株）
TEL：03-5302-1360 FAX：03-5302-1361
E-amil：cp-inq@esol.co.jp

●FPGA開発キット

Spartan-3 Starter Kit

- 20万ゲートのSpartan-3 Platform FPGAである，XC3S200をベースとした開発環境を提供．
- XC3S200 FPGA，コンフィグレーション用PROMおよび1MバイトSRAMを搭載した開発ボード．
- RS-232-C，VGAおよびPS/2ポートI/Oおよびインターフェースのセット．
- JTAGプログラミング・ケーブルおよびユニバーサル電源をサポート．
- ISE FoundationおよびWebPACKデザイン・ソフトウェアを提供．
- プログラマブル・ロジック・ガイドおよびSpartan-3リソースCD付き．

●価格：$99

■ザイリンクス（株）
TEL：03-5321-7740 FAX：03-5321-7762

●数学ライブラリ

インテル マス・カーネル・ライブラリ7.0/インテル クラスタ・マス・カーネル・ライブラリ7.0

- マス・カーネル・ライブラリは，工学，科学，金融系アプリケーション向けに，高度に最適化されたスレッド・セーフな数学関数群を提供し，Windows版とLinux版を用意．
- クラスタ・マス・カーネル・ライブラリには，マス・カーネル・ライブラリの数学関数に加えて，Linuxクラスタ対応のScnLAPACKが含まれる．
- マス・カーネル・ライブラリには，線形代数（BLAS，LAPACK，DSS），離散フーリエ変換（DFT），PARDISO直接法スパースソルバ，ベクトル・マス・ライブラリ（VML），ベクトル・スタティスティカル・ライブラリ（VSL）乱数ジェネレータが含まれる．
- インテル社のPentium 4，Xeon，Itanium 2の各プロセッサに対して最適化．

●価格：¥29,400
（インテルマス・カーネル・ライブラリ7.0）
¥36,540
（インテル クラスタ・マス・カーネル・ライブラリ7.0）

■エクセルソフト（株）
TEL：03-5440-7875 FAX：03-5440-7876
E-mail：xlsoftkk@xlsoft.com

●自動車用リアルタイムOS

RTAプロダクト・ファミリ

- ソフトウェア開発期間の短縮やコストの削減を実現する，OSEK/VDX準拠の自動車用リアルタイムOS．
- RTA-OSEKは，タイミング・パフォーマンス解析ツール（RTA-OSEK Planner），OSEKオペレーティング・システム（RTA-OSEK Component）およびコンフィグレーションを行うツール（RTA-OSEK Builder）で構成．OSEK規格のすべての適合クラスを提供し，MISRA Cに準拠．アプリケーションのリアルタイム性を設計段階で正確にチェックでき，多種のマイコンに対応．
- RTA-TRACEは，自動車用アプリケーションの評価やデバッグをサポートする，分散リアルタイム・トレース環境で，複雑なアプリケーションの挙動をグラフィカルに表示することで，デバッグ作業を効率化．内部OSイベントを容易に把握することで，アプリケーションの信頼性が向上．システムやモジュールを検証するための詳細な統計解析機能をサポートし，多種OSに対応．

●価格：下記へ問い合わせ

■イータス（株）
TEL：045-222-0900
E-mail：sales@etas.co.jp

■弊誌では新製品に関するニュース・リリースを募集しております．
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュース・リリース係
FAX：（03）5395-2127，E-mail：mngnews@cqpub.co.jp （編集部）



INFORMATION
海外・国内イベント/セミナー情報



海外イベント

9/7-11 **ITU TELECOM ASIA 2004**
Busan Exhibition & Convention Center, Busan, Korea
ITU TELECOM
http://www.itu.int/ASIA2004/

9/12-14 **PORTABLE POWER CONFERENCE & EXPO**
Argent Hotel, San Francisco, CA, USA
IDG
http://www.portablepowerconference.com/live/30/

9/21-23 **Wescon**
Anaheim Convention Center, Anaheim, CA, USA
IEEE
http://www.wescon.com/

9/26-10/1 **MobiCom 2004**
Loews Philadelphia Hotel, Philadelphia, PA, USA
ACM SIGMOBILE
http://www.sigmobile.org/mobicom/2004/

10/4-8 **PCB Design Conferences East**
Expo Center of New Hampshire, Manchester, NH, USA
UP Media Group
http://www.pcbeast.com/

10/5-7 **TECHXNY 2004**
Jacob K. Javits Convention Center, New York, NY, USA
CMP Media
http://www.techxny.com/

10/13-16 **electronicAsia 2004**
Hong Kong Convention & Exhibition Centre, Hong Kong, China
Hong Kong Trade Development Council
http://www.electronicasia.com/

国内イベント

9/1-3 2004分析展
幕張メッセ(千葉県千葉市美浜区)
(社)日本分析機器工業会
http://www.jaima.or.jp/show/index.htm

9/15-17 第6回 自動認識総合展
東京ビッグサイト(東京都江東区有明)
(社)日本自動認識システム協会
http://www.autoid-expo.com/

9/22-25 A&Vフェスタ 2004
パシフィコ横浜(神奈川県横浜市西区)
(社)日本オーディオ協会
http://www.jas-audio.or.jp/

9/24-26 東京ゲームショウ 2004
幕張メッセ(千葉県千葉市美浜区)
(社)コンピュータエンターテイメント協会
http://tgs.cesa.or.jp/

9/29-9/30 LinuxWorld C&D/Tokyo 2004
新宿NSビル(東京都新宿区西新宿)
IDGジャパン
http://www.idg.co.jp/expo/lwc/

10/5-9 CEATEC JAPAN
幕張メッセ(千葉県千葉市美浜区)
日本エレクトロニクスショー協会
http://home.jesa.or.jp/jp/exhibitions/ceatec/

10/13-15 第7回 関西 機械要素技術展/設計製造ソリューション展
インテックス大阪(大阪府大阪市住之江区)
リード・エグジビション・ジャパン(株)
http://www.mtech-kansai.jp/
http://www.dms-kansai.jp/

開催日，イベント名，開催地，問い合わせ先の順
日程はすべて予定です．問い合わせ先にご確認のうえ，お出かけください．

セミナー情報

ロボットビジョンセミナー
開催日時　：8月31日(火)/横浜，9月2日(木)/名古屋，9月3日(金)/大阪
開催場所　：横浜情報文化センター(神奈川県横浜市)，名古屋安保ホール(愛知県名古屋市)，大阪千里ライフサイエンスセンター(大阪府豊中市)
受講料　：無料
問い合わせ先：(株)リンクス ロボットビジョンセミナー担当，☎(045)979-0731
http://www.linx.jp/robo-v/

GPL/LGPLセミナー
開催日時　：9月2日(木)～3日(金)
開催場所　：(株)イーエルティ大阪営業所(大阪府大阪市川区宮原)
受講料　：63,000円(税込)
問い合わせ先：(株)イーエルティ，e-mail：seminar@emblit.co.jp
http://www.emblit.co.jp/event_gplsem.html

PC実習!!VB.NETで学ぶオブジェクト指向プログラミング入門
開催日時　：9月9日(木)～10日(金)
開催場所　：SRCセミナールーム(東京都新宿区高田馬場)
受講料　：81,900円(税込)
問い合わせ先：(株)ソフト・リサーチ・センター，☎(03)5272-6071，FAX(03)5272-6345
http://www.src-j.com/seminar_no/24/24_196.htm

UMLモデリング入門～UMLモデリング技能認定試験L1レベル準拠
開催日時　：9月9日(木)～10日(金)
開催場所　：大阪トレーニングルーム(大阪府豊中市)
受講料　：42,000円(税込)
問い合わせ先：(株)オージス総研，☎(03)5440-4771，e-mail：Info_otc@ogis-ri.co.jp
http://www.ogis-ri.co.jp/otc/training/index.html

C言語によるはじめてのLinuxプログラミング
開催日時　：9月9日(木)～10日(木)
開催場所　：エイチアイ研修室(東京都目黒区東山)
受講料　：92,000円(税込，テキスト代を含む)
問い合わせ先：(株)エイチアイICP事業部，☎(03)3719-8155，FAX(03)5773-8661
http://icp.hicorp.co.jp/seminar/linux/clinux.asp

シミュレータ環境構築テクニカルトレーニング(中級)
開催日時　：9月10日(金)
開催場所　：ガイオ・テクノロジー日本橋事業所セミナールーム(東京都中央区日本橋)
受講料　：無料
問い合わせ先：ガイオ・テクノロジー(株)，e-mail：seminar@gaio.co.jp
http://www.gaio.co.jp/event/regular_seminar.html

Linux Pスレッドプログラミング
開催日時　：9月13日(月)
開催場所　：エイチアイ研修室(東京都目黒区東山)
受講料　：49,000円(税込，テキスト代を含む)
問い合わせ先：(株)エイチアイICP事業部，☎(03)3719-8155，FAX(03)5773-8661
http://icp.hicorp.co.jp/seminar/linux/posix_thread.asp

～ISO14764に基づく～ソフトウェア保守の問題解決とその取り組み法
開催日時　：9月14日(火)
開催場所　：SRCセミナールーム(東京都新宿区高田馬場)
受講料　：50,400円(税込)
問い合わせ先：(株)ソフト・リサーチ・センター，☎(03)5272-6071，FAX(03)5272-6345
http://www.src-j.com/seminar_no/24/24_211.htm

～短納期・高品質ソフトウェア開発のための～
テスト駆動開発手法技術解説とその進め方
開催日時　：9月14日(火)～15日(水)
開催場所　：SRCセミナールーム(東京都新宿区高田馬場)
受講料　：79,800円(税込)
問い合わせ先：(株)ソフト・リサーチ・センター，☎(03)5272-6071，FAX(03)5272-6345
http://www.src-j.com/seminar_no/24/24_214.htm

シミュレータファミリ体験コース(入門コース)
開催日時　：9月17日(金)
開催場所　：ガイオ・テクノロジー日本橋事業所セミナールーム(東京都中央区日本橋)
受講料　：無料
問い合わせ先：ガイオ・テクノロジー(株)，e-mail：seminar@gaio.co.jp
http://icp.hicorp.co.jp/seminar/linux/clinuxgui.asp

最先端のソフトウェアテスト
～基礎から最先端，現場での実際，効率的方法まで～
開催日時　：9月17日(金)
開催場所　：オームビル(東京都千代田区神田錦町)
受講料　：83,990円(税込)
問い合わせ先：(株)トリケップス，☎(03)3294-2547，FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/040917a.htm

VC++&Win32APIによるマルチスレッドプログラミング
開催日時　：9月27日(月)
開催場所　：エイチアイ研修室(東京都目黒区東山)
受講料　：49,000円(税込，テキスト代を含む)
問い合わせ先：(株)エイチアイICP事業部，☎(03)3719-8155，FAX(03)5773-8661
http://icp.hicorp.co.jp/seminar/c-vc/vc_multi.asp

～移動通信のための～ディジタル変復調技術入門
開催日時　：9月27日(月)～28日(火)
開催場所　：オームビル(東京都千代田区神田錦町)
受講料　：71,925円(税込/1口，1口で1社3名まで受講可)
問い合わせ先：(株)トリケップス，☎(03)3294-2547，FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/c040927b.htm

DSPの基礎技術
開催日時　：9月30日(木)～10月1日(金)
開催場所　：中央大学駿河台記念館(東京都千代田区神田)
受講料　：65,625円(税込/1口，1口で1社3名まで受講可)
問い合わせ先：(株)トリケップス，☎(03)3294-2547，FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/c040930b.htm

読者の広場

Interface への声



2004年8月号特集「新世代 TRON アーキテクチャ T-Engine 誕生」に関して

▷ TRON アーキテクチャに関する関心があったので，今回の特集はよかったです．職場では TRON の経験者が活躍しています．

以前の号で二足歩行ロボットの特集があったので，次は認識，学習に対する技術を特集してください．　　（万年失業家）

アンケートの結果

興味のあった記事（2004年8月号で実施）

①プロローグ リアルタイム OS の現状
②第1章 T-Engine の思想
③第3章 リアルタイム OS T-Kernel の詳細
④第2章 T-Engine ハードウェアの概要
⑤第7章 T-Kernel と Linux のハイブリッド環境による T-Linux の実装
⑥第5章 T-Engine のミドルウェア
⑦第6章 Windows CE と T-Kernel の協調動作の原理
⑧第4章 T-Engine 開発キットと Teacube
⑨IP パケットの隙間から
⑩プログラミングの要（第14回）
⑪やり直しのための信号数学（第26回，最終回）
⑫Appendix1 ルネサスマイコン搭載 T-Engine のラインナップ
⑬開発技術者のためのアセンブラ入門（第26回）
⑭シニアエンジニアの技術草子（参拾弐拾段）
⑮フリーソフトウェア徹底活用講座（第16回）
⑯TECHNO-FRONTIER 2004
⑰ハッカーの常識的見聞録
⑱Networld + Interop 2004 Las Vegas
⑲Engineering Life in Silicon Valley
⑳組み込みプログラミング・ノウハウ入門（第17回）
㉑C++ による DSP オブジェクト指向プログラミング（第6回）
㉒In-terGiga No.33
㉓Appendix2 ARM9 を2個搭載したマルチ CPU T-Engine

特集『新世代 TRON アーキテクチャ T-Engine 誕生』についてのアンケートの結果

Q1　TRON 系 OS を使って開発を行ったことがありますか？

①はい，今現在使って開発を行っている（0%）
②はい，以前使ったことがある（25%）
③いいえ（75%）

Q2　T-Engine に期待することは何ですか？（複数回答可）

①ミドルウェアの流通（30%）
②統一された開発プラットホームの普及（30%）
③製品に転用できるプラットホーム（25%）
④T-Kernel により大規模開発が可能になること（15%）
⑤その他（0%）

Q3　T-Engine に関連した内容で，どんな記事を希望しますか？（複数回答可）

①T-Engine ハードウェアを使った開発事例（40%）
②デバイス・ドライバの作成（30%）
③T-Kernel を用いたアプリケーション開発（30%）
④その他（0%）



読者プレゼント



●応募方法：本誌読者アンケートはがきに必要事項を記入のうえ，2004年9月30日（必着）までにご応募ください．なお当選者の発表は発送をもってかえさせていただきます．

（1）CD ケース　（1名）
Altera 社



特集担当デスクから

☆一口に“USB”といっても，現在ではその応用範囲は多岐にわたります．USB が登場した頃は，USB のホストは PC 向けチップセットや PCI ベースのホスト・コントローラだけで，「USB 機器の開発」といえば USB ターゲット，つまり PC の周辺機器開発を指していた時代もありました．しかし現在では，USB ホスト機能をもった組み込み機器や，自身がターゲットにもホストにもなる機器など，さまざまな場面で使われるようになりました．
☆ USB ターゲット機器は，仕様的にも比較的簡単であることから，OS を採用せず，USB コントローラを直接制御するプログラムをいきなり書き始めても，完成までもっていくことは難しくないでしょう．しかしホスト機能を実装するとなると，いきなり敷居は高くなります．
☆組み込み機器であれば，仕様としてはじめから「ハブ非対応」，「○○社製 USB キーボードのみ対応」といったように，対応機器を限定する方法もあるのでしょうが，やはり利用する側からすれば，「ハブ対応」，「各社 HID 標準キーボード&マウス対応」，「標準マスストレージ対応」などを期待/要求するでしょう．
☆ホスト/ターゲット両方の機能を内蔵したデバイスで行くのか，On-The-Go デュアル・ロール・デバイスで行くのか，USB 機器を設計するときに，どんなデバイスを採用するかは重要な問題です．今回の特集がデバイス決定の際に参考になればと思います．

Interface 次号予告

2004年11月号は
9月25日発売です

技術者のためのデータ計測

計測/精度/有効数字/補正/補間/蓄積/センシング/システム設計

センサで捕らえたデータは，そのままA-D変換して使えることのほうが少なく，得られたデータを目的にあわせて抽出して計算し，目的に沿った形式へと加工しなければならない．そして数値を計算する際には，有効数字の考え方や誤差の補正などが必要になってくる．もっとも重要なのは，「何を計測したいのか」をはっきりと認識し，正確に数値を収集することである．そのためには，測ろうとしている対象を十分に理解し，余計な数値を拾ってしまわないようにしなければならない．

次号の特集では，データの収集方法，そして得られたデータをいかに目的のために利用していくのか，その点に焦点をあて，解説を行う．そして実際に稼動しているデータ計測システムの考え方と概要を紹介する．

編集後記

●アテネ・オリンピックが開幕した．警備予算は10億ユーロ（約1360億円）超だという．対テロ対策のための費用だ．しかしどのような最新装備をもってしても自爆テロのようなものから安全でいるのは容易ではない．条約や国際法を踏みにじって戦争を始めた強国がいばる．いばるから，いつまでも平和が訪れない．（檀）

●この時期は筆者の会社が夏休みに入るので大変です．最悪，校正期間と重なる場合は，筆者校正を筆者のご自宅に送ることも…．さらに時間がないときはFAXで…．しかし，自宅にはFAXがない場合も！ そんときゃ何らかの方法で電子ファイル化…．しまいにゃスキャナ取り込みで巨大な画像ファイルをメールで送りつけ…（^^;）（M）

●（Y2）さんと同じく，懸賞に当たってしまいました！ 当たったのは，現在B級映画マニアの間で話題沸騰中の作品「いかレスラー」の監督と出演者直筆サイン入りパンフレット．「無我」の西村修は好きなレスラーなので，直筆のサインには感動．サインを前にマントラを唱えて結跏趺坐…はとてもできないけど．（＝Ю）

●なぜプログラマはドキュメントを書かないのか．それは面倒だからであり，どうせ書いてもプログラムの変更に追従できないことが目に見えているからだ．解決策としては，プログラム中にドキュメントを埋め込む，Wikiなどでドキュメント書きのコストを下げるなどが考えられるが，もっと画期的な方法はないのだろうか…．（み）

●よく利用する鉄道の路線に，一部2階建ての車両が登場．2階建て車両は2階部分に人気が集まりがちだが，実は1階のほうが楽しい．座るとホームの高さと目線が同じなので，歩いていく人の脚や靴，こびりついた汚いガムが間近に!! 走行中は線路やすれ違う電車の車輪なんかが迫って来て，ちょっとスリリングだ．（もみ）

●遂に懸賞が当たりました！ いままで地道に応募していたのですが，なかなか当たらないし，もう応募するのもめんどうだから止めようとも思っていたんですが，諦めかけたら当たるなんて，なんだか笑っちゃいます．でも，いったいどのくらいの倍率から勝ち残った（？）んでしょう．ちょっと知りたいです．（Y2）

●1歳の子供が「新幹線チャチャ」という言葉を連発する．何とも可愛い響きである．新幹線の絵柄の水筒のことで中にはお茶が入っているのだ．のどが乾くと寝ている時に急に起きて泣きながら叫ぶのだ．夜中に起こされるのはたまらなくつらいが，愛らしい言葉の響きに思わずにっこりだ．（ちゃん）

●先日田舎に遊びに行った折，ついつい夕方まで半袖でうろついていたら，腕に小さな黒い虫が．気づいた時にはもう遅く，5か所ほどブヨに噛まれていました．翌日からものすごいかゆみに一週間のたうちまわり，すっかり田舎暮らしに慣れて免疫のついた家族を横目に，都会暮らしのもろさを実感した夏でした．（だ）

お知らせ

■ 読者の広場

本誌に関するご意見・ご希望などを，綴じ込みのハガキでお寄せください．読者の広場への掲載分には粗品を進呈いたします．なお，掲載に際しては表現の一部を変更させていただくことがありますので，あらかじめご了承ください．

■ 投稿歓迎

本誌に投稿をご希望の方は，連絡先（自宅/勤務先）を明記のうえ，テーマ，内容の概要をレポート用紙1～2枚にまとめて「Interface投稿係」までご送付ください．メールでお送りいただいても結構です（送り先はsupportinter@cqpub.co.jpまで）．追って採否をお知らせいたします．なお，採用分には小社規定の原稿料をお支払いいたします．

■ 本誌掲載記事についてのご注意

本誌掲載記事には著作権があり，示されている技術には工業所有権が確立されている場合があります．したがって，個人で利用される場合以外は，所有者の許諾が必要です．また，掲載された回路，技術，プログラムなどを利用して生じたトラブルについては，小社ならびに著作権者は責任を負いかねますので，ご了承ください．

本誌掲載記事をCQ出版（株）の承諾なしに，書籍，雑誌，Webといった媒体の形態を問わず，転載，複写することを禁じます．

■ コピー・サービスのご案内

本誌バックナンバの掲載記事については，在庫（原則として24か月分）のないものに限りコピー・サービスを行っています．コピー体裁は雑誌見開きの，複写機による白黒コピーです．なお，コピーの発送には多少時間がかかる場合があります．

- コピー料金（税込み）
  1ページにつき100円
- 発送手数料（判型に関わらず）
  1～10ページ：100円，11～30ページ：200円，31～50ページ：300円，51～100ページ：400円，101ページ以上：600円
- 送付金額の算出方法
  総ページ数×100円＋発送手数料
- 入金方法
  現金書留か郵便小為替による郵送
- 明記事項
  雑誌名，年月号，記事タイトル，開始ページ，総ページ数
- 宛て先
  〒170-8461 東京都豊島区巣鴨1-14-2
  CQ出版株式会社 コピー・サービス係
  （TEL：03-5395-4211，FAX：03-5395-1642）

■ お問い合わせ先のご案内

- 在庫，バックナンバ，年間購読送付先変更に関して
  販売部：03-5395-2141
- 広告に関して
  広告部：03-5395-2133
- 雑誌本文に関して
  編集部：03-5395-2122

記事内容に関するご質問は，返信用封筒を同封して編集部宛てに郵送してくださるようお願いいたします．筆者に回送してお答えいたします．

Interface
©CQ出版（株） 2004 振替 00100-7-10665
2004年10月号 第30巻 第10号（通巻第328号）
2004年10月1日発行（毎月1日発行）
定価は裏表紙に表示してあります

発行人／増田久喜
編集人／山形孝雄
編集／大野典宏 村上真紀 山口光樹 大竹友美 小林由美子
デザイン・DTP／クニメディア株式会社
表紙デザイン／株式会社プランニング・ロケッツ
本文イラスト／森 祐子
広告／澤辺 彰 中元正夫 菅原利江

発行所／CQ出版株式会社 〒170-8461 東京都豊島区巣鴨1-14-2
電話／編集部（03）5395-2122 FAX／（03）5395-2127
広告部（03）5395-2133 URL http://www.cqpub.co.jp/interface/
販売部（03）5395-2141 E-mail supportinter@cqpub.co.jp
CQ Publishing Co.,Ltd.／1-14-2 Sugamo, Toshima-ku, Tokyo 170-8461, Japan
印刷／クニメディア株式会社 美和印刷株式会社
製本／星野製本株式会社


日本ABC協会加盟誌
（新聞雑誌部数公査機構） ISSN0387-9569